08-02-00

A

07/31/00 JC760 U.S. PTO

Please type a plus sign (+) inside this box → [+]

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

# UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

| | |
|---|---|
| Attorney Docket No. | SUN-P4677 |
| First Inventor or Application Identifier | Richard Burridge |
| Title | Method and Apparatus for Collocating... |
| Express Mail Label No. | EL575422743US |

JC886 U.S. PTO 09/629080 07/31/00

**APPLICATION ELEMENTS**
See MPEP chapter 600 concerning utility patent application contents.

**ADDRESS TO:** Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

1. [ ] * Fee Transmittal Form (e.g., PTO/SB/17) (Submit an original and a duplicate for fee processing)
2. [X] Specification [Total Pages 40] (preferred arrangement set forth below)
   - Descriptive title of the Invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings (if filed)
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) (35 U.S.C. 113) [Total Sheets 24]
4. Oath or Declaration [Total Pages 4]
   a. [X] Newly executed (original or copy)
   b. [ ] Copy from a prior application (37 C.F.R. § 1.63(d)) (for continuation/divisional with Box 16 completed)
      i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(a) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

*NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

5. [ ] Microfiche Computer Program (Appendix)
6. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
   a. [ ] Computer Readable Copy
   b. [ ] Paper Copy (identical to computer copy)
   c. [ ] Statement verifying identity of above copies

**ACCOMPANYING APPLICATION PARTS**

7. [X] Assignment Papers (cover sheet & document(s))
8. [ ] 37 C.F.R.§3.73(b) Statement (when there is an assignee) [X] Power of Attorney
9. [ ] English Translation Document (if applicable)
10. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
11. [ ] Preliminary Amendment
12. [X] Return Receipt Postcard (MPEP 503) (Should be specifically itemized)
13. [ ] * Small Entity Statement(s) (PTO/SB/09-12) [ ] Statement filed in prior application, Status still proper and desired
14. [ ] Certified Copy of Priority Document(s) (if foreign priority is claimed)
15. [ ] Other: ____________

16. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment:
[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No: ______/______
Prior application information: Examiner ____________ Group / Art Unit: ____________

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

**17. CORRESPONDENCE ADDRESS**

[ ] Customer Number or Bar Code Label (Insert Customer No. or Attach bar code label here) or [ ] Correspondence address below

| | | | | | |
|---|---|---|---|---|---|
| Name | David B. Ritchie | | | | |
| | D'Alessandro & Ritchie | | | | |
| Address | P. O. Box 640640 | | | | |
| City | San Jose | State | CA | Zip Code | 95164-0640 |
| Country | USA | Telephone | 408-441-1100 | Fax | 408-441-8400 |

| | | | |
|---|---|---|---|
| Name (Print/Type) | John P. Schaub | Registration No. (Attorney/Agent) | 42,125 |
| Signature | [illegible] | Date | 7/31/00 |

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

00073 08052500

SUN-P4677

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

JC886 U.S. PTO
09/629080
07/31/00

| | |
|---|---|
| In re Application of: | ) Art Unit: |
| Burridge et al. | ) Examiner: |
| Serial No. [Not yet assigned] | ) |
| Filed: July 31, 2000 | ) |
| For: METHOD AND APPARATUS FOR COLLOCATING DYNAMICALLY LOADED PROGRAM FILES | ) |

09629080.073100

CERTIFICATE OF MAILING
"Express Mail" mailing label no: EL575422743US
Date of Deposit: July 31, 2000
I hereby certify that this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to:
Box Patent Application
Assistant Commissioner for Patents
Washington, D.C. 20231

Diane Morse
Diane Morse

## TRANSMITTAL LETTER

Honorable Assistant Commissioner
for Patents
Box Patent Application
Washington, D.C. 20231

Dear Sir:

Enclosed for filing please find the patent application for an invention entitled, "METHOD AND APPARATUS FOR COLLOCATING DYNAMICALLY LOADED PROGRAM FILES", filed on behalf of Sun Microsystems, Inc., assignee from inventors Richard Burridge and Jeffrey Kesselman, including Utility Patent Application

Transmittal, 29 pages of specification, 10 pages of claims, 24 sheets of drawing figures, and 1 pages of Abstract. Also enclosed herewith are the Declaration, Power of Attorney.

The attorney's Docket Number is SUN-P4677.

Kindly address all communications regarding this application to:

David B. Ritchie
D'Alessandro & Ritchie
P.O. Box 640640
San Jose, CA 95164-0640
Telephone (408) 441-1100

A check in the amount of $1,524.00 is enclosed for the filing fee for a large entity, calculated as follows:

| | |
|---|---|
| Basic Filing Fee: | $ 690.00 |
| 7 Additional Independent Claim: | $ 546.00 |
| 16 Additional Dependent Claims: | $ 288.00 |
| Total Filing Fee: | $1,524.00 |

In the event any variance exists between the amount enclosed and the Patent Office charges for filing the above-noted documents, the Assistant Commissioner is hereby authorized to charge or credit the difference to our Deposit Account No. 04-0025.

A duplicate of this page is enclosed.

Respectfully submitted,
D'ALESSANDRO & RITCHIE

Dated: July 31, 2000

[illegible]

John P. Schaub
Reg. No. 42,125

D'Alessandro & Ritchie
P.O. Box 640640
San Jose, CA 95164-0640
(408) 441-1100

**This application is submitted in the name of inventors Richard Burridge and Jeffrey Kesselman, assignors to Sun Microsystems, Inc., a Delaware Corporation.**

S P E C I F I C A T I O N

# METHOD AND APPARATUS FOR COLLOCATING DYNAMICALLY LOADED PROGRAM FILES

## BACKGROUND OF THE INVENTION

### Field Of the Invention

The present invention relates to object-oriented computer systems having classes that are dynamically loaded at runtime. More particularly, the present invention relates to a method and apparatus for collocating dynamically loaded program files.

### Background

Programming languages may be classified according to the time at which program units required for program execution are loaded. Dynamic languages allow program units to be loaded dynamically as they are needed. Static languages require that all program units required for program execution be loaded prior to program execution. The "late binding" provided by dynamic languages means that programs only grow as large as they need to be at runtime. Additionally, dynamic languages also enhance program modularity because dynamic languages make fewer compiled-in assumptions about the implementation of data structures than static, early-binding languages like C or C++.

Figure 1 is a block diagram that illustrates dynamically loaded program files. A loader 10 first loads the main program unit. Program execution proceeds until a program unit that has not already been loaded is referenced. At this point, the loader 10 loads the referenced program unit contained in a file. The loader 10 loads the files on an as needed basis. Dynamically loading program files is described in more detail with reference to Fig. 1A.

Turning now to Fig. 1A, a flow diagram that illustrates dynamically loading program files is presented. At reference numeral 400, a runtime loader is invoked to execute the main program unit. When the runtime loader encounters a reference to a program unit that has not already been loaded (405), the runtime loader searches known input sources until either the program unit is found, or until all input sources have been exhausted (410). The runtime loader loads the program unit (420) if it is found (415).

The dynamic language program files loaded may be categorized many ways. The files loaded are frequently categorized as either system files 12 or application files 14. The term "system files" 12 typically refers to files containing program units supplied as part of a particular execution environment. Conversely, the term "application files" 14 typically refers to files containing program units not supplied as part of a particular execution environment. The system files 12 and application files 14 may be individual program files that are loaded and executed directly. The system files 12 and application files 14 may also be libraries containing one or more individual program files that are first extracted from a library and then loaded.

The Java™ language is one example of a dynamic language. In Java™ technology, a class loader loads classes dynamically one at a time, as they are needed. Any Java™ class can be loaded into a running Java™ interpreter at any time, possibly even over the network. When the Java™ application or applet is loaded in this manner, however, each of the files is transferred in uncompressed form using a separate request, making the process relatively inefficient.

Each Java™ execution environment includes a collection of services and libraries referred to as the Java™ Development Kit (JDK). New applications are created by adding services and libraries that build upon the standard JDK. As these new Java™ services and libraries are developed, their functionality is typically provided to developers as a set of packages provided in different JAR (Java™ ARchive) files, which are archives of files used by a Java™ application. A JAR file allows compressed versions of one or more files to be combined into a single JAR file. Each JAR file also includes a "manifest" file that contains meta-information about the files in the archive.

To use a JAR file, the JAR file is specified as the value of the archive attribute of an applet or application:

```
<APPLET archive="sound.jar" code="Soundmap.class" width=288
height=288>
...
</APPLET>
```

The “code” attribute tells the browser which of the class files in the archive contains the main method to be executed. The “archive” attribute specifies where to look for files. The archive attribute may specify a list of JAR files. The Web browser or applet viewer searches these archives for any files the applet requires. If a file is not found in an archive, the browser attempts to load the file from the Web using a new HTTP request.

Java™ applications wishing to use new services and libraries must ensure the appropriate JAR files are available to the application at runtime. This requirement is made difficult by the fact that JAR files may be located in multiple places and on multiple servers. Each JAR file may also contain extra Java™ classes or packages that the Java™ application does not use but which are automatically loaded as a part of application initialization.

Development of an application program typically involves creating multiple program units that reference other program units. Each referenced program unit may in turn reference other program units. In a dynamic language environment, each referenced program unit must be loaded at run time before the referenced unit is used. Loading more program units than necessary decreases execution efficiency. Often, removing the reference to the program unit may prevent loading a program unit at run time. For example, a user or developer may create a local method to perform a task, rather than invoke an external program unit. Accordingly, a need exists in the prior art for a method and apparatus for determining which program units are loaded during execution of a dynamically loaded program.

Additionally, loading program units that are unneeded and loading uncompressed program units increases startup time and increases the memory footprint of the running application. Accordingly, a further need exists in the prior art for a method and apparatus for collocating only the program units required for executing a dynamic language program.

## SUMMARY OF THE INVENTION

A method for executing a dynamically loaded program having a main program unit includes executing the main unit a first time, creating at least one library file containing only application program files loaded during the first execution, specifying a system program file input and executing the main program unit a second time using the system program file input and the at least one library file for dynamically loaded program files. A method for optimizing a dynamically loaded program, the program including a main program unit includes creating at least one library file containing only application program files loaded during execution of the main program unit and optimizing the program based upon a list of application program files in the library file. A method for testing a dynamically loaded program, the program including a main program unit includes specifying a list including at least one application program file to be tested, creating at least one library file containing only application program files loaded during execution of the main program unit and indicating incomplete test coverage when at least one file in the list is not represented in the library file.

## BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram that illustrates dynamically loaded program files.

Fig. 1A is a flow diagram that illustrates dynamically loading program files.

Fig. 2A is a block diagram that illustrates one embodiment of the present invention.

Fig. 2B is a high level flow diagram that illustrates creating and using an optimized library file in accordance with one embodiment of the present invention.

Fig. 2C is a flow diagram that illustrates dynamically loading program files in accordance with one embodiment of the present invention.

Fig. 3A is a block diagram that illustrates creating an optimized library file containing application program files in accordance with one embodiment of the present invention.

Fig. 3B is a block diagram that illustrates executing a dynamically loaded program in accordance with one embodiment of the present invention.

Fig. 4 is a flow diagram that illustrates executing a dynamically loaded program in accordance with one embodiment of the present invention.

Fig. 5 is a flow diagram that illustrates creating a library file in accordance with one embodiment of the present invention.

Fig. 6 is a flow diagram that illustrates storing loaded application program files to a library file in accordance with one embodiment of the present invention.

Fig. 7A is a flow diagram that illustrates optimizing a dynamically loaded program in accordance with one embodiment of the present invention.

Fig. 7B is a flow diagram that illustrates performing optimizations in accordance with one embodiment of the present invention.

Fig. 8 is a flow diagram that illustrates creating a library file in accordance with one embodiment of the present invention.

Fig. 9 is a flow diagram that illustrates testing a dynamically loaded program in accordance with one embodiment of the present invention.

Fig. 10A is a block diagram that illustrates one embodiment of the present invention.

Fig. 10B is a high level flow diagram that illustrates creating and using an optimized JAR file in accordance with one embodiment of the present invention.

Fig. 10C is a block diagram that illustrates creating a Java™ ARchive (JAR) file containing class files in accordance with one embodiment of the present invention.

Fig. 11A is a block diagram that illustrates executing a dynamically loaded Java™ application in accordance with one embodiment of the present invention.

Fig. 11B is a block diagram that illustrates a JAR file representation in accordance with one embodiment of the present invention.

Fig. 12 is a flow diagram that illustrates executing a dynamically loaded Java™ application in accordance with one embodiment of the present invention.

Fig. 13 is a flow diagram that illustrates creating a JAR file in accordance with one embodiment of the present invention.

Fig. 14 is a flow diagram that illustrates storing loaded application class files to a JAR file in accordance with one embodiment of the present invention.

Fig. 15A is a flow diagram that illustrates optimizing a Java™ application in accordance with one embodiment of the present invention.

Fig. 15B is a flow diagram that illustrates performing optimizations in accordance with one embodiment of the present invention.

## DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Those of ordinary skill in the art will realize that the following description of the present invention is illustrative only. Other embodiments of the invention will readily suggest themselves to such skilled persons having the benefit of this disclosure.

This invention relates to object-oriented computer systems. More particularly, the present invention relates to a method and apparatus for collocating dynamically loaded program files. The invention further relates to machine readable media on which are stored (1) the layout parameters of the present invention and/or (2) program instructions for using the present invention in performing operations on a computer. Such media includes by way of example magnetic tape, magnetic disks, optically readable media such as CD ROMs and semiconductor memory such as PCMCIA cards. The medium may also take the form of a portable item such as a small disk, diskette or cassette. The medium may also take the form of a larger or immobile item such as a hard disk drive or a computer RAM.

According to one embodiment of the present invention, application program files required for execution of a dynamically loaded program are collocated in a library file. Subsequent invocations of the dynamically loaded program retrieve application program files from the library file. The present invention allows increased execution efficiency and decreases application program size by collocating only the application program files required during execution of the program.

Although the program execution environment described herein is described with reference to Java™ technology, the invention has a broader scope. The invention could include any program execution environment featuring dynamically loaded program files. The present invention may be applied to executing, optimizing and testing any program in such a program execution environment.

Since this disclosure is described with reference to the Java™ programming language, this description will utilize the nomenclature of Java™. The following Java™ nomenclature is used frequently throughout the description and will be described herein briefly. A class is a grouping of instance variables and methods that is used to describe the behavior of an object. An object is an instance of a class.

A method is a program segment that performs a well-defined series of operations. In Java™ technology, a method is implemented by instructions represented as a stream of bytecodes. A bytecode is an 8-bit code that can be a portion of an instruction such as an 8-bit operand or opcode. An interface is an abstract class where the bytecodes that implement the method are defined at runtime. A Java™ application is an executable module consisting of bytecodes that can be executed using the Java™ interpreter or the

Java™ just-in-time compiler. A more detailed description of the features of the Java™ programming language is provided in James Gosling, Bill Joy and Guy Steele, *The Java™ Language Specification*, Addison Wesley (1996).

Turning now to Fig. 2A, a block diagram that illustrates one embodiment of the present invention is presented. Figure 2A is an overview illustrating the sequence of steps used to produce a library file containing application program files. A compiler 16 compiles source code 18 to create program files 20. A user or developer determines which main method will be optimized, and specifies the pathnames for application program files 22 and system program files 24.

The goal of the offline loader 26 is to determine which program units are application program units and write them to a library file. The offline loader 26 loads the main method and begins to execute it. If the offline loader 26 requires a program unit that has not already been loaded, the offline loader 26 obtains the program unit from the pathnames specified. If the loaded program unit is an application program unit, the offline loader 26 writes a copy of the program file to an application library file 28 and continues executing the main method. When the main method has terminated, the application library file 28 contains all application program files 22 loaded during the execution of the main method.

Throughout this disclosure, the term "pre-run" is used to denote the process described above, wherein an offline loader executes the main method and creates a library file containing all application files loading during execution of the main method.

As mentioned above, application library file 28 contains all application program files 22 loaded during the execution of the main method. If all possible execution paths were exercised at reference numeral 26, the application library file 28 will contain all the application program files that will be referenced during subsequent executions of the main method. Therefore, according to this embodiment of the present invention, subsequent invocations of the main method specify the application library file 28 as the input source for application program files. Runtime loader 30 is used to load and execute the main method during subsequent executions of the main method.

It should be noted that the method and system described herein pertains to creating a library file containing only application program files. However, the method and system described herein is not limited to these particular types of program files. Any program file grouping could be used using the method and system described herein.

Turning now to Fig. 2B, a high level flow diagram that illustrates creating and using an optimized library file in accordance with one embodiment of the present invention is presented. A user or developer first identifies the main program unit to be optimized (32). The user then specifies the pathname(s) for application files and system program files (34) and invokes an offline loader (36). The offline loader is configured to load program units required by the loader during a first execution of the main program unit. The offline loader is further configured to create a library file, and write to the library file all program units loaded during the first execution of the main program unit (38). Preferably, every execution path is executed during the first execution.

The library file created at reference numeral 38 is used as the input source for application program files in subsequent executions of the main program unit. For each subsequent execution of the main program unit, the library file is specified as the input source for application files (40) and a runtime loader is invoked to execute the main program unit (42). When the runtime loader requires an application program unit not already loaded, the runtime loader obtains the unit from the library file, thus significantly reducing the amount of overhead involved in obtaining application program files from multiple sources. The runtime loader actions are described in more detail with reference to Fig. 2C.

Turning now to Fig. 2C, a flow diagram that illustrates dynamically loading program files in accordance with one embodiment of the present invention is presented. At reference numeral 430, a runtime loader is invoked to execute the main program unit. When the runtime loader encounters a reference to a program unit that has not already been loaded (435), the runtime loader first searches for the program unit. The runtime loader looks first in the previously created library file and a system program file input source (440). Since the library file contains application program files that were loaded during a prior execution of the same main program unit, all application program units referenced during subsequent invocations of the main program unit should be found in the library file. However, if the program unit has not been found after the initial search at reference numeral 440, the runtime loader optionally searches an alternate source for the program unit. The runtime loader loads the program unit (460) if it is found (455).

Turning now to Fig. 3A, a block diagram that illustrates creating an optimized library file containing application program files in accordance with one embodiment of the present invention is presented. During program execution, application program files 46 and system program files 48 are loaded as needed by an offline loader 50. The offline loader 50 checks each program file loaded to determine whether the program file is an application program file 46 or a system program file 48. Application program files 46 are stored to a library file 52.

The description of program files as either application program files 46 or system program files 48 is not intended to be limiting in any way. Those of ordinary skill in the art will recognize that the invention may be applied to other program file groupings.

Turning now to Fig. 3B, a block diagram that illustrates executing a dynamically loaded program in accordance with one embodiment of the present invention is presented. The library file 52 created as illustrated in Fig. 2 serves as the input source for application program files. During program execution, system program files 54 and application program files from at least one library file 56 are loaded as needed by a run time loader 58.

Turning now to Fig. 4, a flow diagram that illustrates executing a dynamically loaded program in accordance with one embodiment of the present invention is presented. At reference numeral 70, a library file containing only application program files loaded during execution of the main program unit is created. At reference numeral 75, pathnames for system program files and the library file created at reference numeral 70 is specified. At reference numeral 80, the main program unit is invoked using the pathnames specified in reference numeral 75 as input sources for system program files 54 and application program files.

Turning now to Fig. 5, a flow diagram that illustrates creating a library file 55 in accordance with one embodiment of the present invention is presented. Figure 5 provides more detail regarding reference numeral 70 in Fig. 4. At reference numeral 100, pathnames for system program files and application program files are specified. At reference numeral 105, the main program unit is invoked. At reference numeral 110, each application program file loaded during execution of the main program unit is written to a library file.

Turning now to Fig. 6, a flow diagram that illustrates storing loaded application program files to a library file in accordance with one embodiment of the present invention is presented. At reference numeral 130, program files are loaded as needed during execution of the main program unit. At reference numeral 135, a determination is made regarding whether a loaded program file is a system program file. If the loaded program file is not a system program file, the file is stored to the library file at reference numeral 140. At reference numeral 145, a determination is made regarding whether execution of the main program unit has terminated. If execution of the main program unit has not been terminated, storing application program files to a library file continues at reference numeral 130.

As mentioned previously, development of an application program typically involves creating multiple program units that reference other program units. Each referenced program unit may in turn reference other program units. In a dynamic language environment, each referenced program unit must be loaded at run time before the referenced unit is used. Loading more program units than necessary decreases execution efficiency. Often, removing the reference to the program unit may prevent loading a program unit at run time. For example, a user or developer may create a local method to perform a task, rather than invoke an external program unit.

According to one embodiment of the present invention, a list of application program files loaded during execution of a dynamically loaded program is used to optimize the application program. Application program files that are not expected to be required may initiate further investigation to determine whether the program files are required. If the unexpected application program files are not required, modifications may be made to other application program files to remove the reference to the unexpected program files, thus decreasing the memory footprint and increasing the execution efficiency of the application program. This embodiment is described further with reference to Figures 7A to 8.

Turning now to Fig. 7A, a flow diagram that illustrates optimizing a dynamically loaded program in accordance with one embodiment of the present invention is presented. At reference numeral 148, an application program is pre-run using an offline loader. At reference numeral 150, the offline loader creates a library file containing only application program files loaded by the offline loader. At reference numeral 152, program optimizations are performed based upon whether the list of program units in the library file is expected.

Turning now to Fig. 7B, a flow diagram that illustrates performing optimizations in accordance with one embodiment of the present invention is presented. Figure 7B corresponds to reference numeral 152 in Fig. 7A. At reference numeral 156, a developer examines the pathname of an application program file included in the library file. If the developer determines pathname is expected (158), the next pathname is examined at reference numeral 156. The presence of an unexpected application program file indicates that during program execution, another program unit (the referencing program unit) referenced the unexpected application program file (the referenced program unit). If the pathname is not expected, the developer removes the reference by first determining the referencing program unit and then modifying referencing program unit to remove the reference (160).

An example of an unexpected pathname is the case where a developer is developing a cohesive set of program units that are supposed to be all located within a certain directory structure. Here, a developer might expect that all application program files loaded during execution of a program under test would have pathnames that shared a common directory path. Noticing an unexpected pathname could initiate further inquiry as to the reason why the program unit was referenced. If it is determined that the reference should be removed, the referencing program unit is modified to remove the reference.

Turning now to Fig. 8, a flow diagram that illustrates creating a library file 30 in accordance with one embodiment of the present invention is presented. Figure 8 provides more detail regarding reference numeral 150 in Fig. 7A. Pathnames for system program files and application program files are specified at reference numerals 170 and 175, respectively. At reference numeral 180, the main program unit is executed. At reference numeral 190, each application program file 35 loaded during execution of the main program unit is stored to the library file 30.

According to another embodiment of the present invention, two application program file lists are compared to determine whether a test plan includes at least one test for each application program file to be tested. A first list includes the application program files to be tested by a test plan. A second list includes the application program files loaded during execution of the test plan for a dynamically loaded program. The first list and the second list are compared. The absence of an application program file in the second list indicates that the test plan failed to include at least one test for the absent application program file. This embodiment is described further with reference to Fig. 9.

Turning now to Fig. 9, a flow diagram that illustrates testing a dynamically loaded program in accordance with one embodiment of the present invention is presented. At reference numeral 200, a list of application program files to be tested is specified. At reference numeral 205, a library file containing only application program files loaded during execution of the main program unit is created. At reference numeral 210, a determination is made regarding whether every file in the list of application program files is represented in the library file. If at least one file is not represented in the list, an indication of incomplete test coverage is made at reference numeral 215. An indication of incomplete test coverage means that the test plan did not result in executing any execution paths within each of the application program files that are represented in the list of application program files to be tested but are not represented in the library file.

According to other embodiments of the present invention, the invention is applied to a Java™ technology environment. According to these embodiments, the application program files are class files, the system program files are part of the Java™ Development Kit (JDK), the loader is a Java™ class loader and the library file is a Java™ ARchive (JAR) file. These embodiments are described further with reference to Figures 10A to 15B.

Turning now to Fig. 10A, a block diagram that illustrates one embodiment of the present invention is presented. Figure 2A is an overview illustrating the sequence of steps used to produce a JAR file containing class files. A compiler 218 compiles source code 216 to create class files 220 A user or developer determines which main method will be optimized, and specifies the pathnames for application class files 222 and system class files 224.

The goal of the offline loader 226 is to determine which class files are application class files and write them to a JAR file 228. The offline class loader 226 loads the main method and begins to execute it. If the offline class loader 226 requires a class file that has not already been loaded, the offline class loader 226 obtains the class file from the pathnames specified. If the loaded class file is an application class file 222, the offline class loader 226 writes a copy of the class file to an application JAR file 228 and continues executing the main method. When the main method has terminated, the application JAR file 228 contains all application class files 222 loaded during the execution of the main method.

As mentioned above, application JAR file 228 contains all application class files 222 loaded during the execution of the main method. If all possible execution paths were exercised at reference numeral 226, the application JAR file 228 will contain all the application class files that will be referenced during subsequent executions of the main method. Therefore, according to this embodiment of the present invention, a subsequent invocations of the main method specify the application JAR file 228 as the input source for application class files. Runtime class loader 300 is used to load and execute the main method during subsequent executions of the main method. At run time, Runtime class loader 300 obtains application class files from the application JAR file 28, thus significantly reducing the amount of overhead involved in obtaining application class files from multiple sources.

Turning now to Fig. 10B, a high level flow diagram that illustrates creating and using an optimized JAR file in accordance with one embodiment of the present invention is presented. A user or developer first identifies the main program unit to be optimized (232). The user then specifies the pathname(s) for application class files and system class files (234) and invokes an offline class loader (236). The offline class loader is configured to load class files required by the class loader during a first execution of the main program unit. The offline class loader is further configured to create a JAR file, and write to the JAR file all class files loaded during the first execution of the main program unit (238).

The JAR file created at reference numeral 238 is used as the input source for application class files in subsequent executions of the main method. For each subsequent execution of the main method, the JAR file is specified as the input source for application class files (240) and a runtime class loader is invoked to execute the main method (242). When the runtime class loader requires an application class file not already loaded, the runtime class loader obtains the class file from the JAR file.

Turning now to Fig. 10C, a block diagram that illustrates creating a Java™ ARchive (JAR) file 246 containing class files in accordance with one embodiment of the present invention is presented. During execution of a Java™ application, application class files 244 and system class files 248 are loaded as needed by an offline Java™ class loader 250. The offline class loader 250 checks each class file loaded to determine whether the class file is an application class file or a system class file. Application class files 244 are stored to a JAR file 246.

The description of a JAR file 246 as a repository for application class files 244 loaded by an offline class loader 250 is not intended to be limiting in any way. Those of ordinary skill in the art will recognize that other file formats may be used.

Moreover, the creation of a single library file or JAR file 246 is not intended to be limiting in any way. The method and system described herein is not limited to producing this one file. Other file configurations can be used including, but not limited to, multiple JAR files 246 containing application program units such as class files.

Turning now to Fig. 11A, a block diagram that illustrates executing a dynamically loaded Java™ application in accordance with one embodiment of the present invention is presented. The JAR file 246 created as illustrated in Fig. 10C serves as the input source for application class files. During execution of an application, system class files 250 and application class files from at least one JAR file 255 are loaded as needed by a runtime class loader 260. The format of a JAR file is described further with respect to Fig. 11B.

Turning now to Fig. 11B, a block diagram illustrating a JAR file representation in accordance with one embodiment of the present invention is presented. The JAR file 262 includes a manifest file 264 and at least one field for storing a class file 266, 268, 270. According to one embodiment of the present invention, the manifest file 264 includes a pathname for the main class file 272. When the runtime class loader is invoked with a JAR file input, the main class file pathname 272 found in the manifest file 264 is used to locate the main method, which is then loaded and executed. Additionally, application class files are loaded from the JAR file 262 during execution of the main method.

Turning now to Fig. 12, a flow diagram that illustrates executing a dynamically loaded Java™ application in accordance with one embodiment of the present invention is presented. At reference numeral 274, a JAR file containing only application class files loaded during execution of the main method is created. At reference numeral 276, pathnames for the JDK and a JAR file are specified. At reference numeral 280, the main method is invoked, using the pathnames specified in reference numeral 276 as input sources for system class files and application class files.

Turning now to Fig. 13, a flow diagram that illustrates creating a JAR file in accordance with one embodiment of the present invention is presented. Figure 13 provides more detail regarding reference numeral 274 in Fig. 12. At reference numeral 300, pathnames for the JDK and application class files are specified. At reference numeral 305, the main method is invoked. At reference numeral 310, each application class file loaded during execution of the main method is written to a JAR file.

Turning now to Fig. 14, a flow diagram that illustrates storing loaded application class files to a JAR file in accordance with one embodiment of the present invention is presented. Figure 14 provides more detail regarding reference numeral 310 in Fig. 13. At reference numeral 320, class files are loaded as needed during execution of the main method. At reference numeral 325, a determination is made regarding whether a loaded class file is a system class file. If the loaded class file is not a system class file, the class file is stored to a JAR file 255. At reference numeral 335, a determination is made regarding whether execution of the main method has terminated. If execution of the main method has not terminated, storing application class files to a JAR file continues at reference numeral 320.

According to one embodiment of the present invention, a list of application class files loaded during execution of a dynamically loaded program is used to optimize the application program. Application class files that are not expected to be required may initiate further investigation to determine whether the program files are required. If the unexpected application program files are not required, modifications may be made to other application class files to remove the reference to the unexpected class files, thus decreasing the memory footprint and increasing the execution efficiency of the application program. This embodiment is described further with reference to Figures 15A to 15B.

Turning now to Fig. 15A, a flow diagram that illustrates optimizing a Java™ application in accordance with one embodiment of the present invention is presented. At reference numeral 340, an application program is pre-run using an offline class loader. At reference numeral 345, the offline class loader creates a JAR file containing only application class files loaded by the offline class loader. At reference numeral 350, program optimizations are performed based upon whether the list of class files in the JAR file is expected.

Turning now to Fig. 15B, a flow diagram that illustrates performing optimizations in accordance with one embodiment of the present invention is presented. Figure 15B corresponds to reference numeral 350 in Fig. 15A. At reference numeral 360, a developer examines the pathname of an application class file included in the JAR file. If the developer determines pathname is expected (365), the next pathname is examined at reference numeral 360. The presence of an unexpected application class file indicates that during program execution, another program unit (the referencing program unit) referenced the unexpected application program unit (the referenced program unit). If the pathname is not expected, the developer removes the reference by first determining the referencing program unit and then modifying referencing program unit to remove the reference (370).

Although the method and system described herein has been described with reference to the Java™ programming language, it is applicable to computer systems using other object-oriented classes that utilize dynamic runtime loading of classes.

According to a presently preferred embodiment, the present invention may be implemented in software or firmware, as well as in programmable gate array devices, Application Specific Integrated Circuits (ASICs), and other hardware.

Thus, a novel method and apparatus for collocating dynamically loaded program files has been described. While embodiments and applications of this invention have been shown and described, it would be apparent to those skilled in the art having the benefit of this disclosure that many more modifications than mentioned above are possible without departing from the inventive concepts herein. The invention, therefore, is not to be restricted except in the spirit of the appended claims.

## CLAIMS

What is Claimed is:

1. A method for executing a dynamically loaded program, said program including a main program unit, said method comprising:

executing said main program unit a first time;

creating at least one library file containing only application program files loaded during said first execution of said main program unit;

specifying a system program file input; and

executing said main program unit a second time using said system program file input and said at least one library file for dynamically loaded program files.

2. The method of claim 1 wherein said creating further comprises:

specifying a first at least one pathname for system program files;

specifying a second at least one pathname for application program files;

executing said main program unit using said first at least one pathname and said second at least one pathname for dynamically loaded program files; and

storing each application program file loaded during execution of said main program unit to said library file.

3. The method of claim 2 wherein said storing further comprises:

loading a program file when referenced during execution of said main program unit;

storing said program file to said library file when said program file is an application program file; and

determining whether execution of said main program unit has terminated.

4. The method of claim 3 wherein said library file further comprises a compressed file.

5. The method of claim 1 wherein said program files comprise Java™ class files and Java™ archive files.

6. The method of claim 5 wherein said system program file input comprises the Java™ Development Kit.

7. The method of claim 6 wherein said library file comprises a Java™ archive file.

8. A method for optimizing a dynamically loaded program, said program including a main program unit, said method comprising:

creating at least one library file containing only application program files loaded during execution of said main program unit; and

optimizing said program based upon a list of application program files in said library file.

9. The method of claim 8 wherein said creating further comprises:

specifying a first at least one pathname for system program files;

specifying a second at least one pathname for application program files;

executing said main program unit using said first at least one pathname and said second at least one pathname for dynamically loaded program files; and

storing each application program file loaded during execution of said main program unit to said library file.

10. The method of claim 9 wherein said storing further comprises:

loading a program file when referenced during execution of said main program unit;

storing said program file to said library file when said program file is an application program file; and

determining whether execution of said main program unit has terminated.

11. The method of claim 10 wherein said library file further comprises a compressed file.

12. The method of claim 8 wherein said program files comprise Java™ class files and Java™ archive files.

13. The method of claim 12 wherein said system program file input comprises the Java™ Development Kit.

14. The method of claim 13 wherein said library file comprises a Java™ archive file.

15. The method of claim 13 wherein said optimizing further comprises:

receiving an application program pathname of a referenced program unit in said library file;

determining a referencing program unit when said pathname is unexpected, said referencing program unit referencing said referenced program unit; and

modifying said referencing program unit to remove any reference to said referenced unit.

16. A method for testing a dynamically loaded program, said program including a main program unit, said method comprising:

specifying a list including at least one application program file to be tested;

creating at least one library file containing only application program files loaded during execution of said main program unit; and

indicating incomplete test coverage when at least one file in said list is not represented in said library file.

17. The method of claim 16 wherein said creating further comprises:

specifying a first at least one pathname for system program files;

specifying a second at least one pathname for application program files;

executing said main program unit using said first at least one pathname and said second at least one pathname for dynamically loaded program files; and

storing each application program file loaded during execution of said main program unit to said library file.

18. The method of claim 17 wherein said storing further comprises:

loading a program file when referenced during execution of said main program unit;

storing said program file to said library file when said program file is an application program file; and

determining whether execution of said main program unit has terminated.

19. The method of claim 18 wherein said library file further comprises a compressed file.

20. The method of claim 16 wherein said program files comprise Java™ class files and Java™ archive files.

21. The method of claim 20 wherein said system program file input comprises the Java™ Development Kit.

22. The method of claim 21 wherein said library file comprises a Java™ archive file.

23. An apparatus for executing a dynamically loaded program, said program including a main program unit, said apparatus comprising:
means for executing said main program unit a first time;
means for creating at least one library file containing only application program files loaded during said first execution of said main program unit;
means for specifying a system program file input; and
means for executing said main program unit a second time using said system program file input and said at least one library file for dynamically loaded program files.

24. An apparatus for optimizing a dynamically loaded program, said program including a main program unit, said apparatus comprising:
means for creating at least one library file containing only application program files loaded during execution of said main program unit; and
means for optimizing said program based upon a list of application program files in said library file.

25. An apparatus for testing a dynamically loaded program, said program including a main program unit, said apparatus comprising:
means for specifying a list including at least one application program file to be tested;
means for creating at least one library file containing only application program files loaded during execution of said main program unit; and

means for indicating incomplete test coverage when at least one file in said list is not represented in said library file.

26. A program storage device readable by a machine, embodying a program of instructions executable by the machine to perform a method to execute a dynamically loaded program, the method comprising:

executing said main program unit a first time;

creating at least one library file containing only application program files loaded during said first execution of said main program unit;

specifying a system program file input; and

executing said main program unit a second time using said system program file input and said at least one library file for dynamically loaded program files.

27. The program storage device of claim 26 wherein said creating further comprises:

specifying a first at least one pathname for system program files;

specifying a second at least one pathname for application program files;

executing said main program unit using said first at least one pathname and said second at least one pathname for dynamically loaded program files; and

storing each application program file loaded during execution of said main program unit to said library file.

28. The program storage device of claim 27 wherein said storing further comprises:

loading a program file when referenced during execution of said main program unit;

storing said program file to said library file when said program file is an application program file; and

determining whether execution of said main program unit has terminated.

29. The program storage device of claim 26 wherein said library file further comprises a compressed file.

30. The program storage device of claim 26 wherein said program files comprise Java™ class files and Java™ archive files.

31. The program storage device of claim 30 wherein said system program file input comprises the Java™ Development Kit.

32. The program storage device of claim 31 wherein said library file comprises a Java™ archive file.

33. A program storage device readable by a machine, embodying a program of instructions executable by the machine to perform a method to optimize a dynamically loaded program, the method comprising:

creating at least one library file containing only application program files loaded during execution of said main program unit; and

optimizing said program based upon a list of application program files in said library file.

34. A program storage device readable by a machine, embodying a program of instructions executable by the machine to perform a method to test a dynamically loaded program, the method comprising:

specifying a list including at least one application program file to be tested;

creating at least one library file containing only application program files loaded during execution of said main program unit; and

indicating incomplete test coverage when at least one file in said list is not represented in said library file.

35. A method for representing a library file, said method including:

storing in at least one program unit field the pathname of every program unit loaded during the execution of a dynamically loaded program, said dynamically loaded program including a main program unit; and

storing in a main unit field the pathname of said main program unit.

36. The method of claim 35 wherein

said program unit field said main unit field are contained within a JAR file and

said main unit field comprises a manifest file.

## ABSTRACT OF THE DISCLOSURE

A method for executing a dynamically loaded program having a main program unit includes executing the main unit a first time, creating at least one library file containing only application program files loaded during the first execution, specifying a system program file input and executing the main program unit a second time using the system program file input and the at least one library file for dynamically loaded program files. A method for optimizing a dynamically loaded program, the program including a main program unit includes creating at least one library file containing only application program files loaded during execution of the main program unit and optimizing the program based upon a list of application program files in the library file. A method for testing a dynamically loaded program, the program including a main program unit includes specifying a list including at least one application program file to be tested, creating at least one library file containing only application program files loaded during execution of the main program unit and indicating incomplete test coverage when at least one file in the list is not represented in the library file.

EL575422743US

SUN-P4677

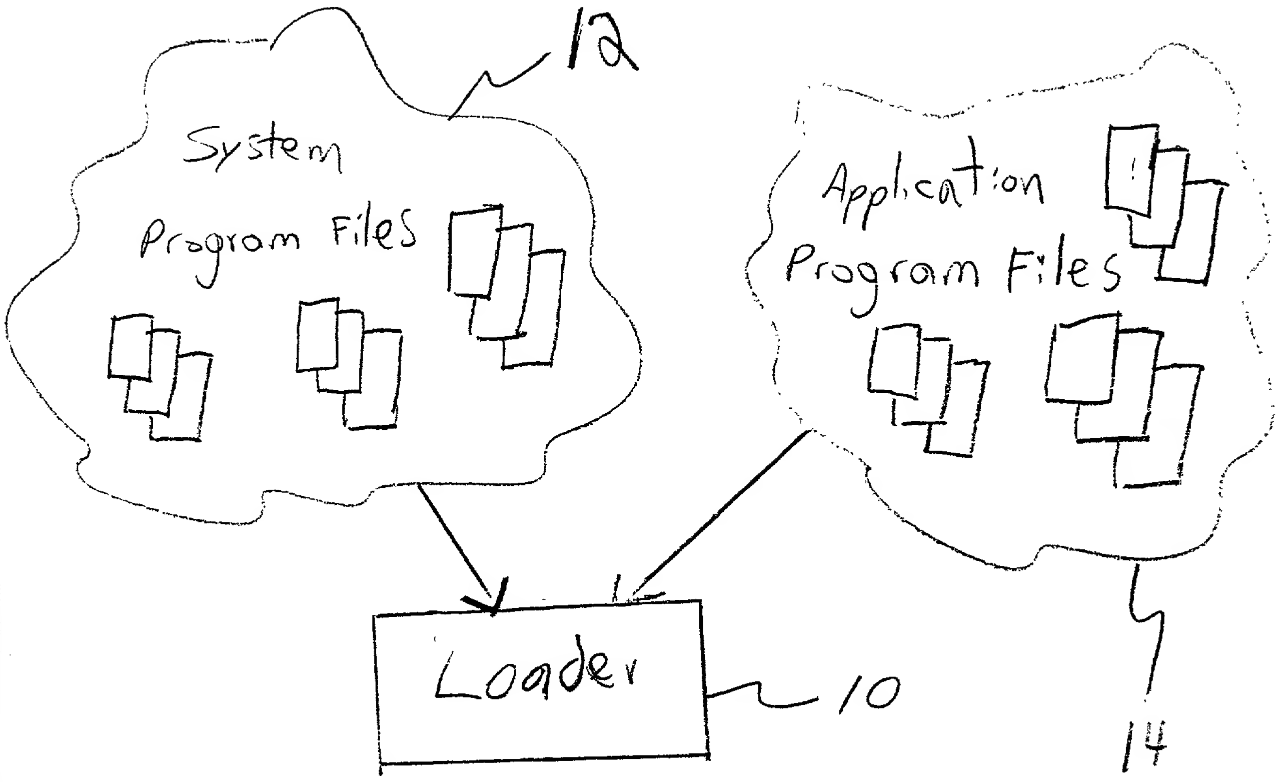
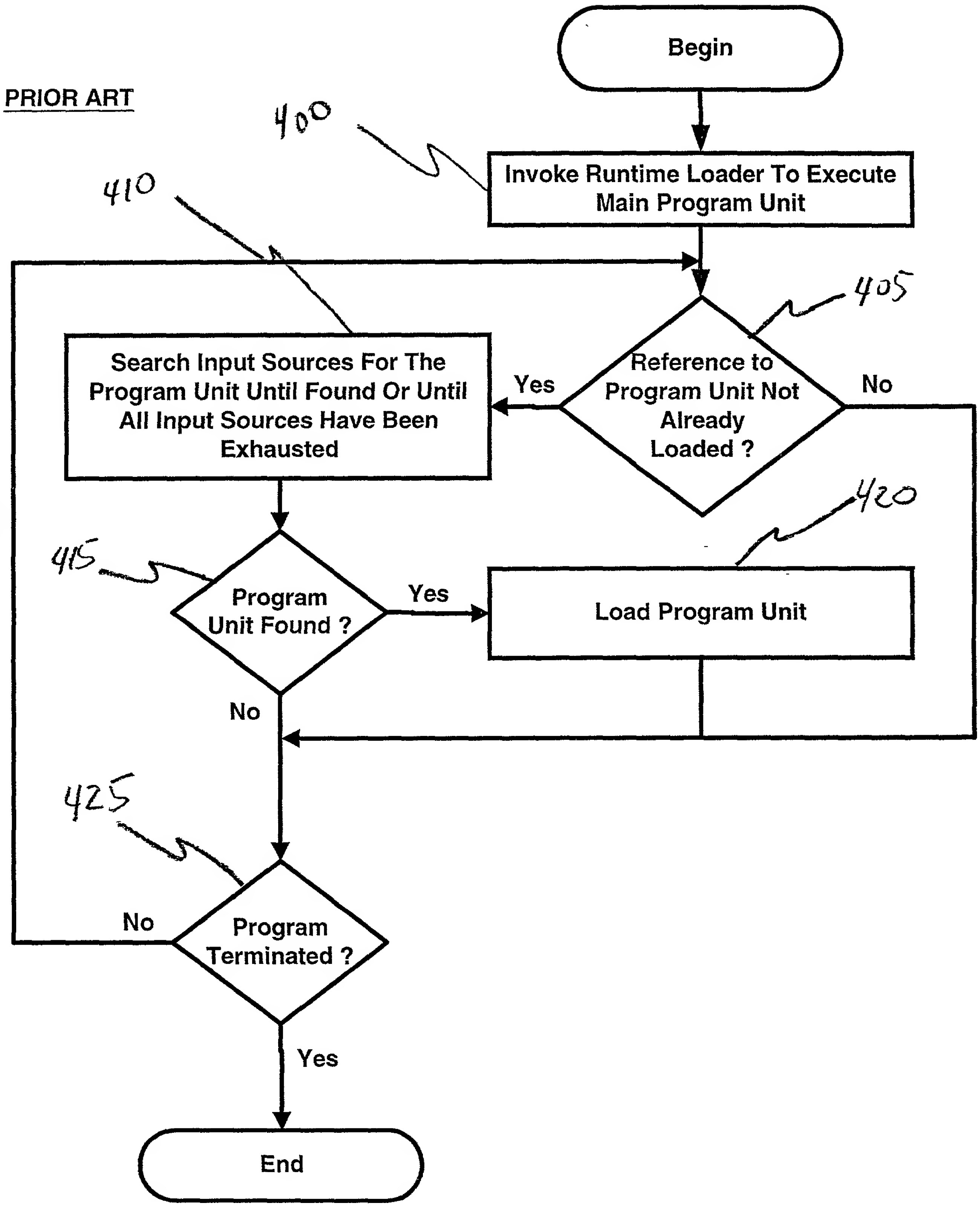


Fig. 1

PRIOR ART



Begin
400
Invoke Runtime Loader To Execute Main Program Unit
410
405
Search Input Sources For The Program Unit Until Found Or Until All Input Sources Have Been Exhausted
Yes
Reference to Program Unit Not Already Loaded ?
No
420
415
Program Unit Found ?
Yes
Load Program Unit
No
425
No
Program Terminated ?
Yes
End

Fig. 1A

EL575422743US

SUN-P4677

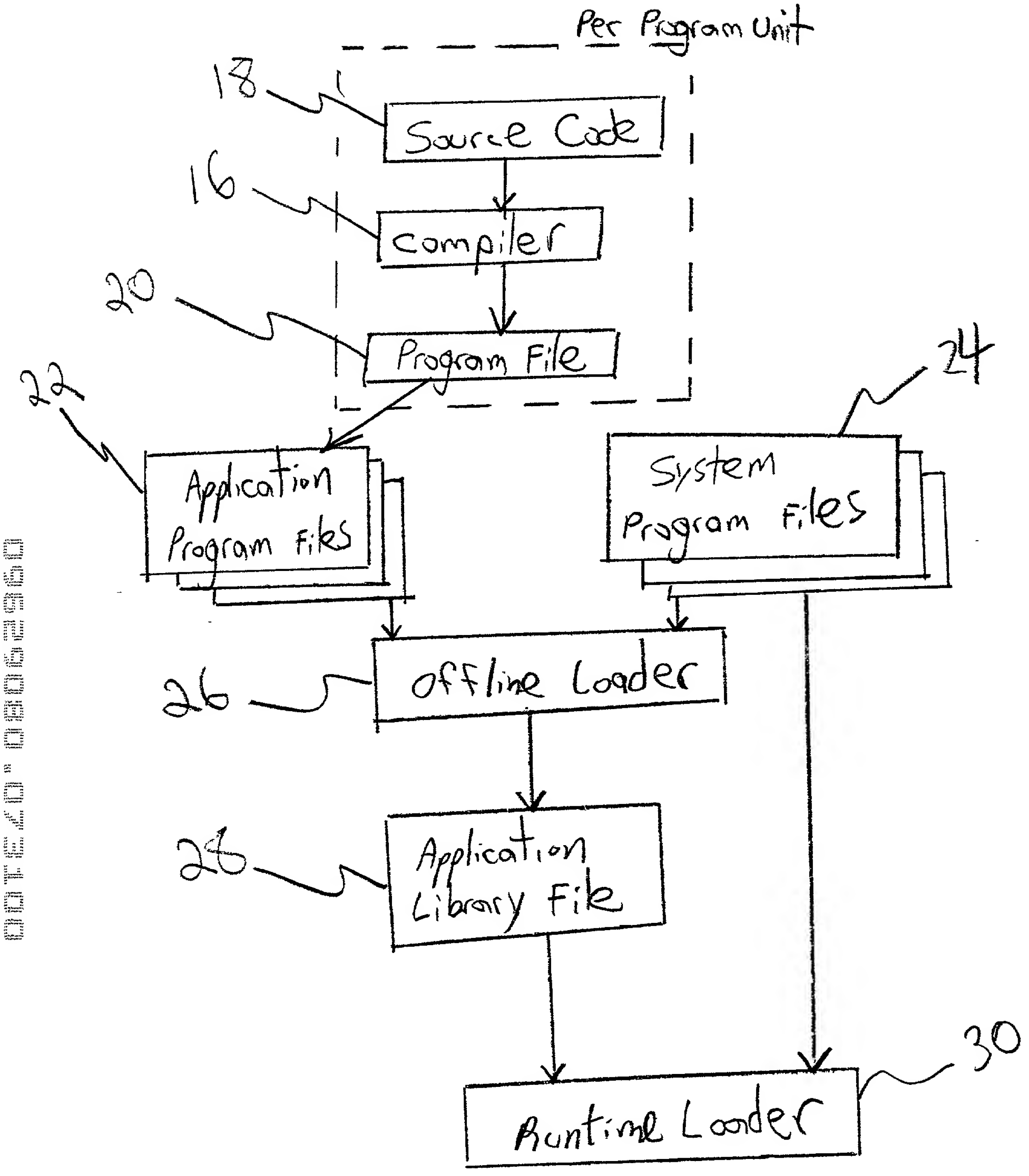


Fig. 2A

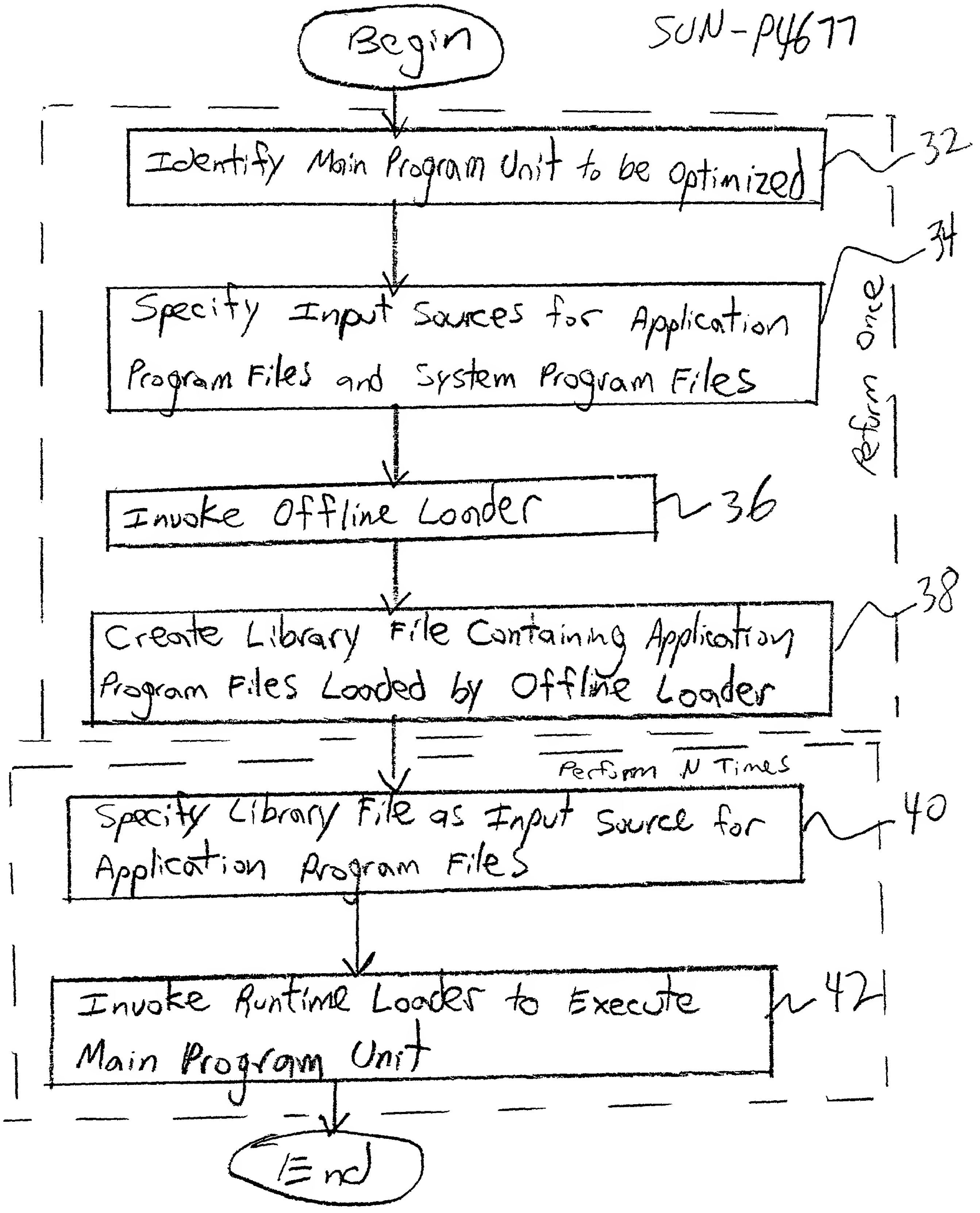

Begin
Identify Main Program Unit to be optimized
32
Specify Input Sources for Application Program Files and System Program Files
34
Perform Once
Invoke Offline Loader
36
Create Library File Containing Application Program Files Loaded by Offline Loader
38
Perform N Times
Specify Library File as Input Source for Application Program Files
40
Invoke Runtime Loader to Execute Main Program Unit
42
End

Fig. 2B

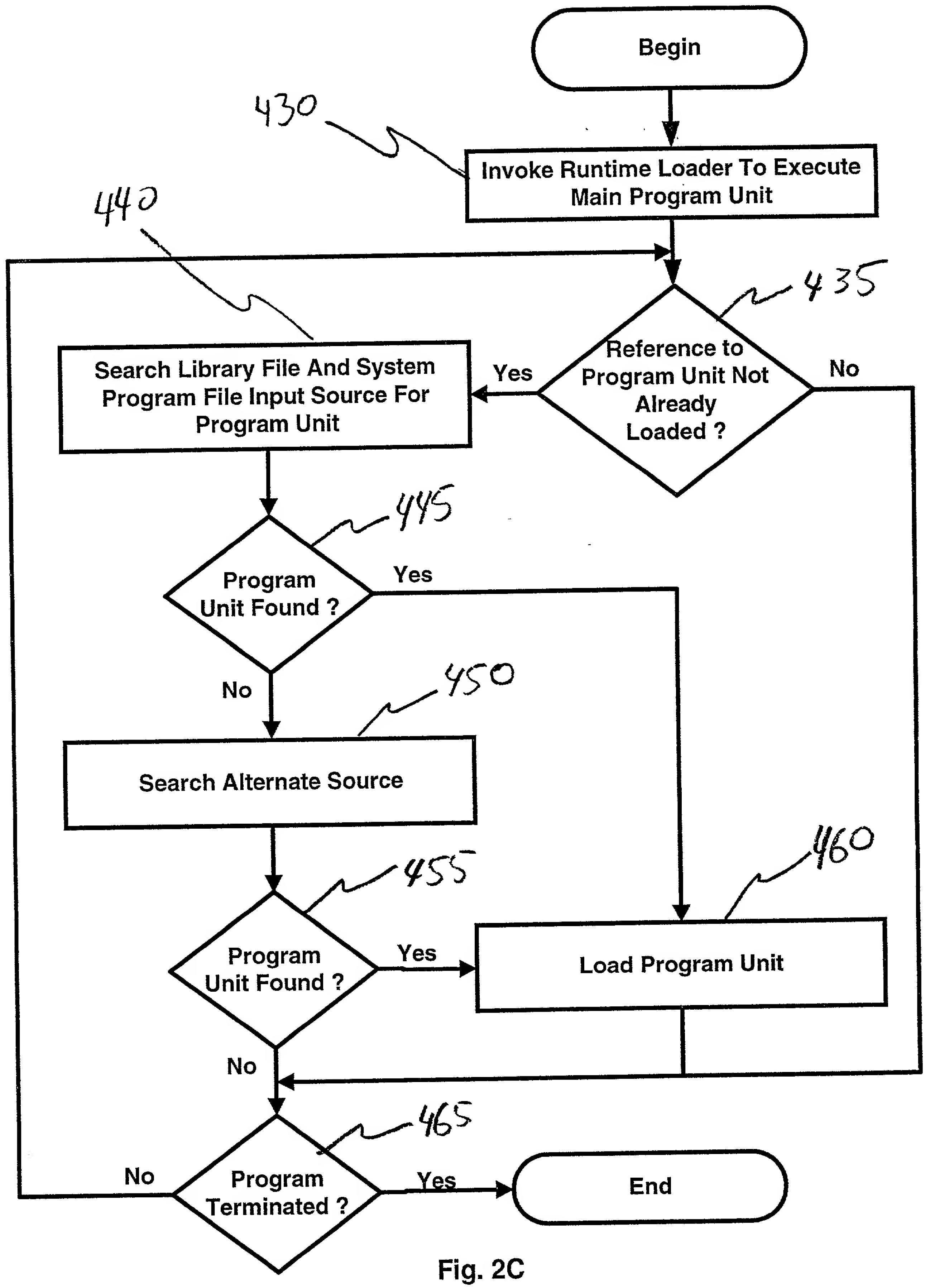

Begin
430
Invoke Runtime Loader To Execute Main Program Unit
440
435
Reference to Program Unit Not Already Loaded ?
Yes
No
Search Library File And System Program File Input Source For Program Unit
445
Program Unit Found ?
Yes
No
450
Search Alternate Source
455
Program Unit Found ?
Yes
No
460
Load Program Unit
465
Program Terminated ?
No
Yes
End

Fig. 2C

EL575422743US

SUN-P4677

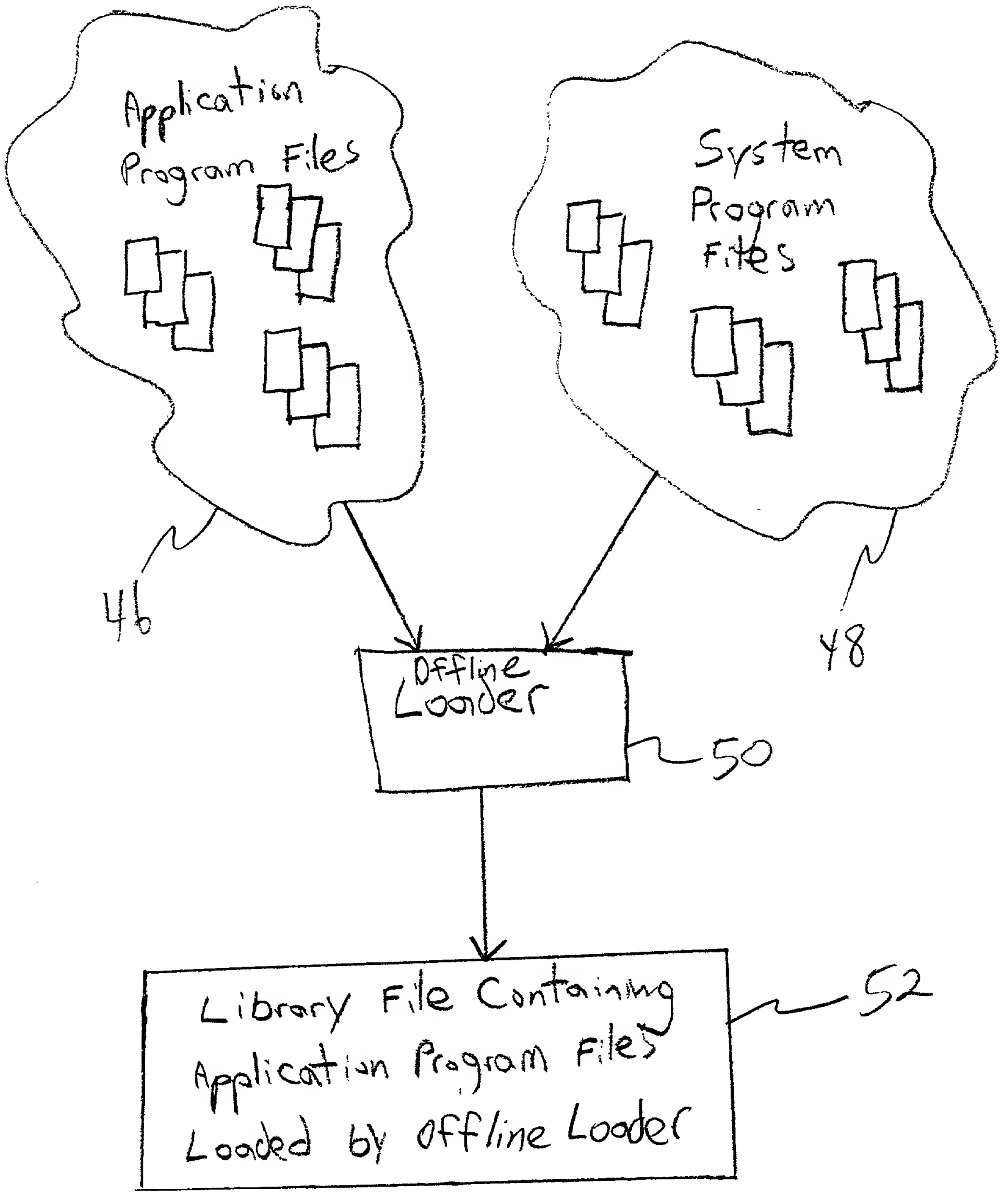

Application Program Files
System Program Files
46
48
Offline Loader
50
Library File Containing Application Program Files Loaded by Offline Loader
52

Fig. 3A

EL575422743US

SUN-P4677

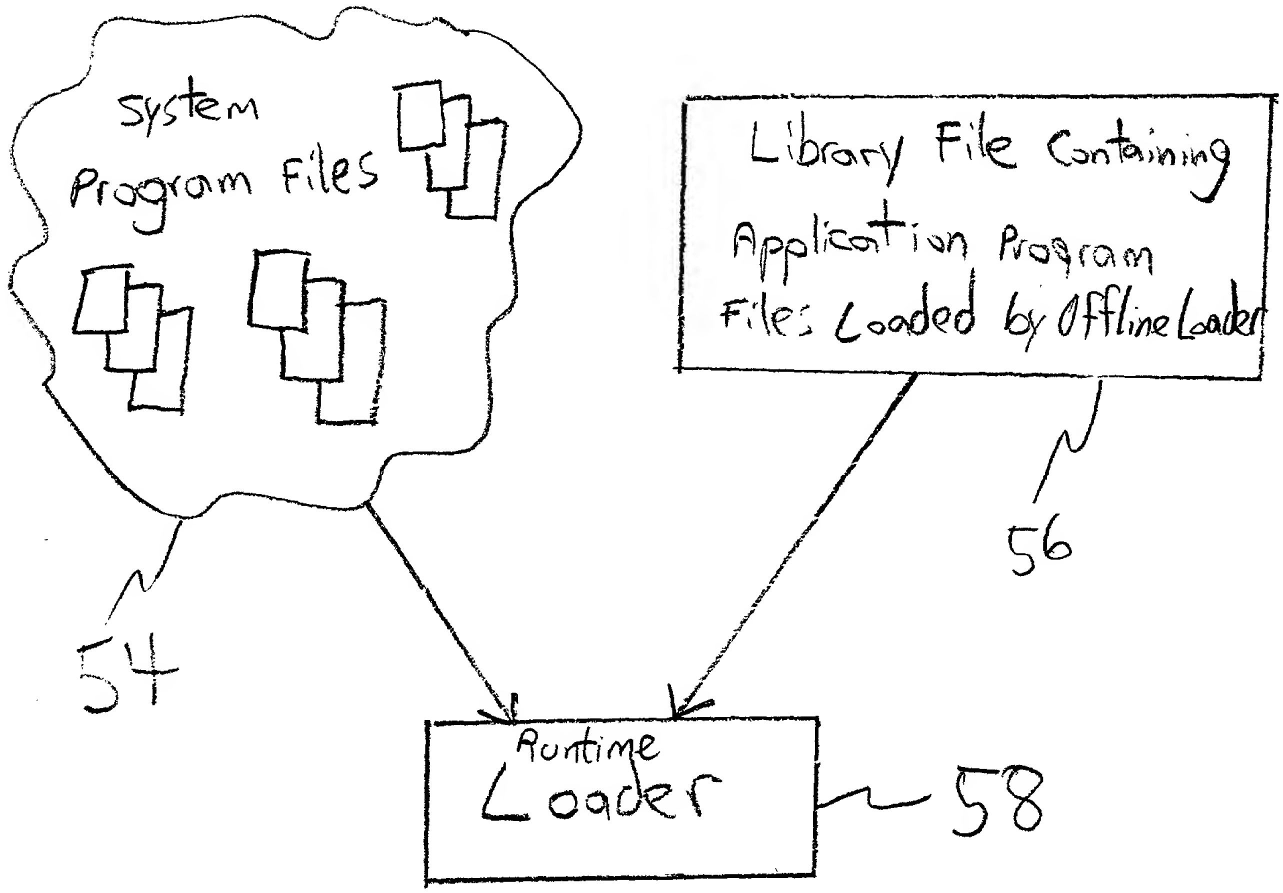
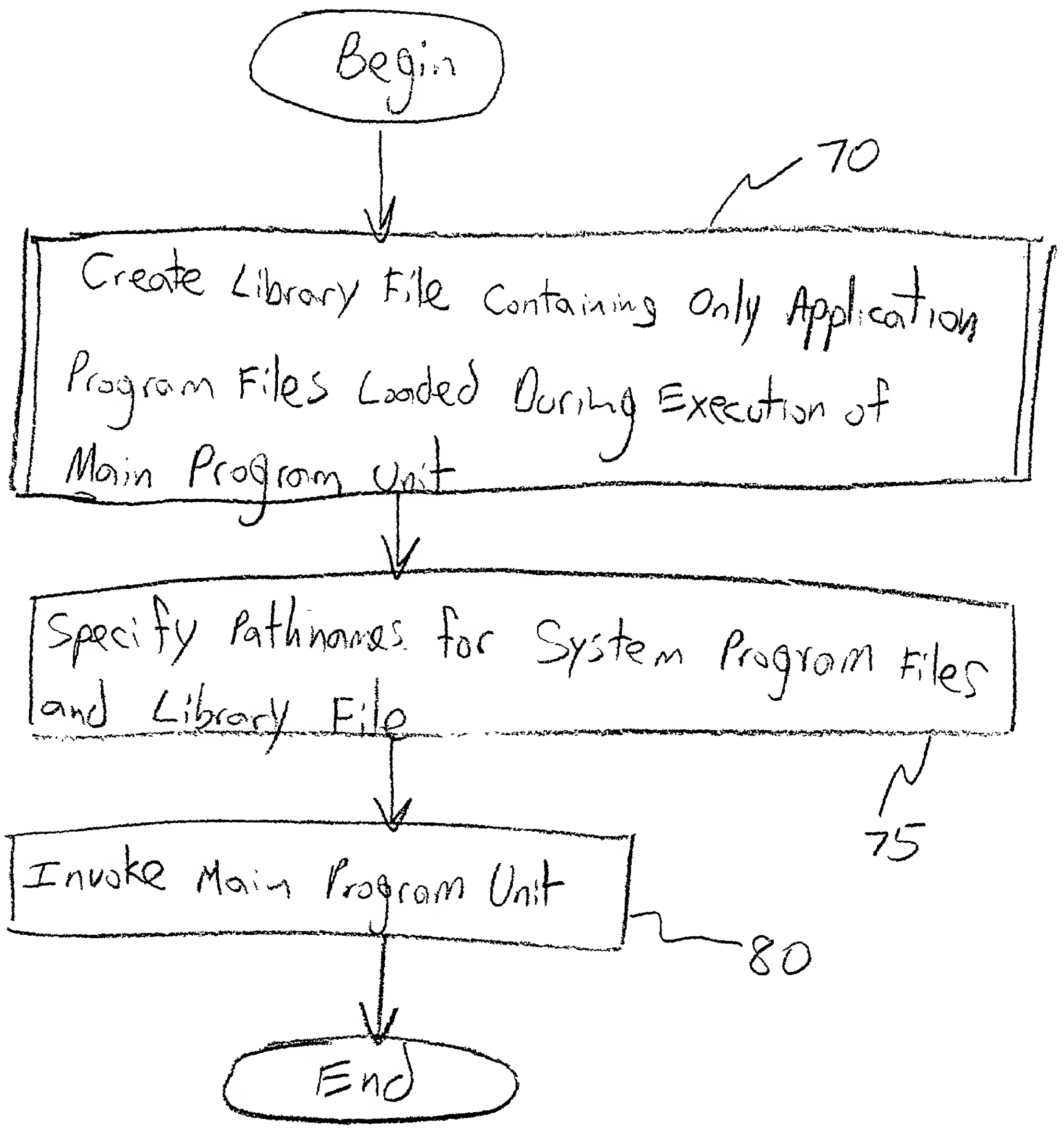

System Program Files
Library File Containing Application Program Files Loaded by Offline Loader
56
54
Runtime Loader
58

Fig. 3B

EL575422743US

SUN-P4677

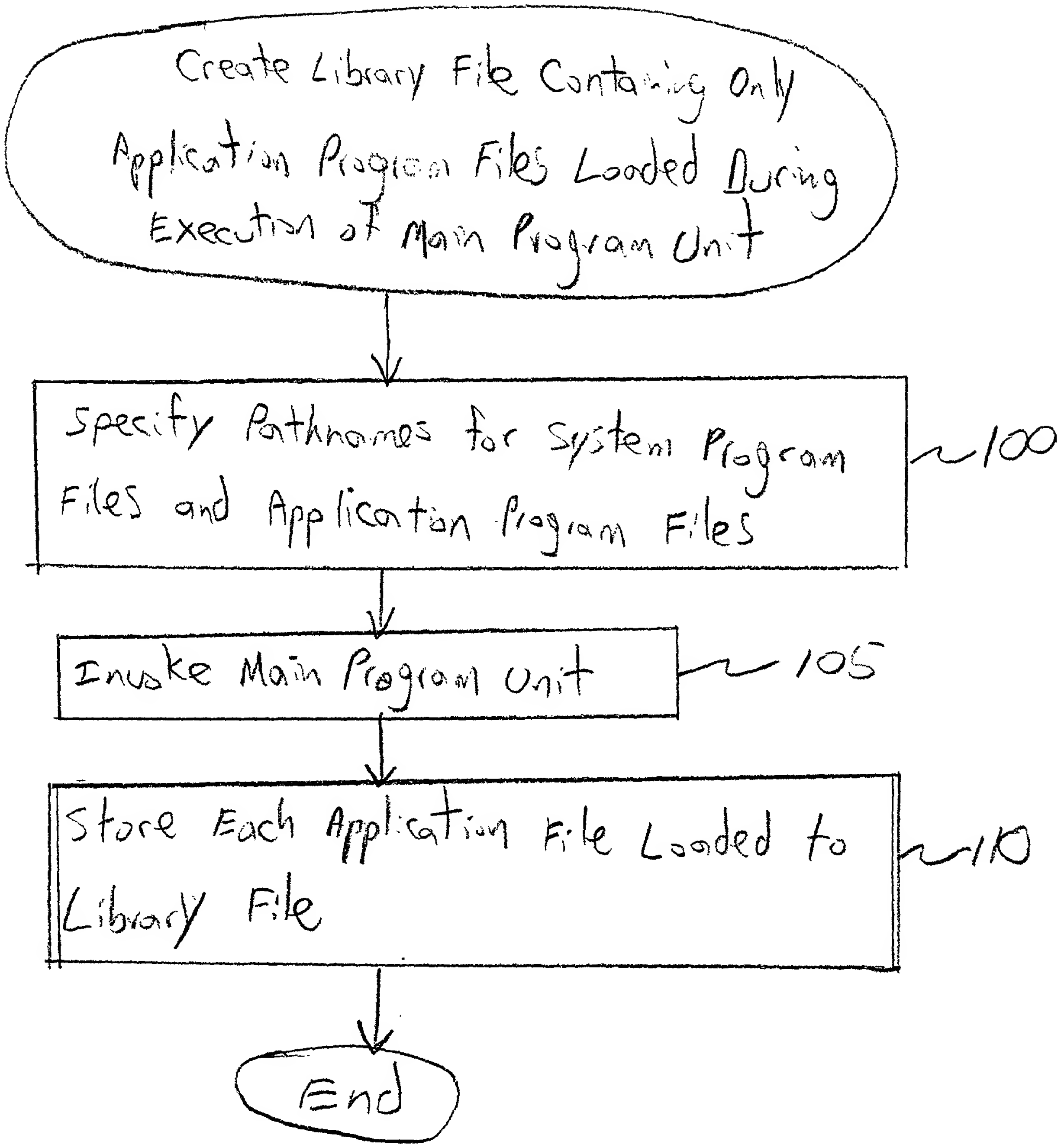


Begin
70
Create Library File Containing Only Application Program Files Loaded During Execution of Main Program Unit
Specify Pathnames for System Program Files and Library File
75
Invoke Main Program Unit
80
End

Fig. 4

EL575422743US

SUN-P4677

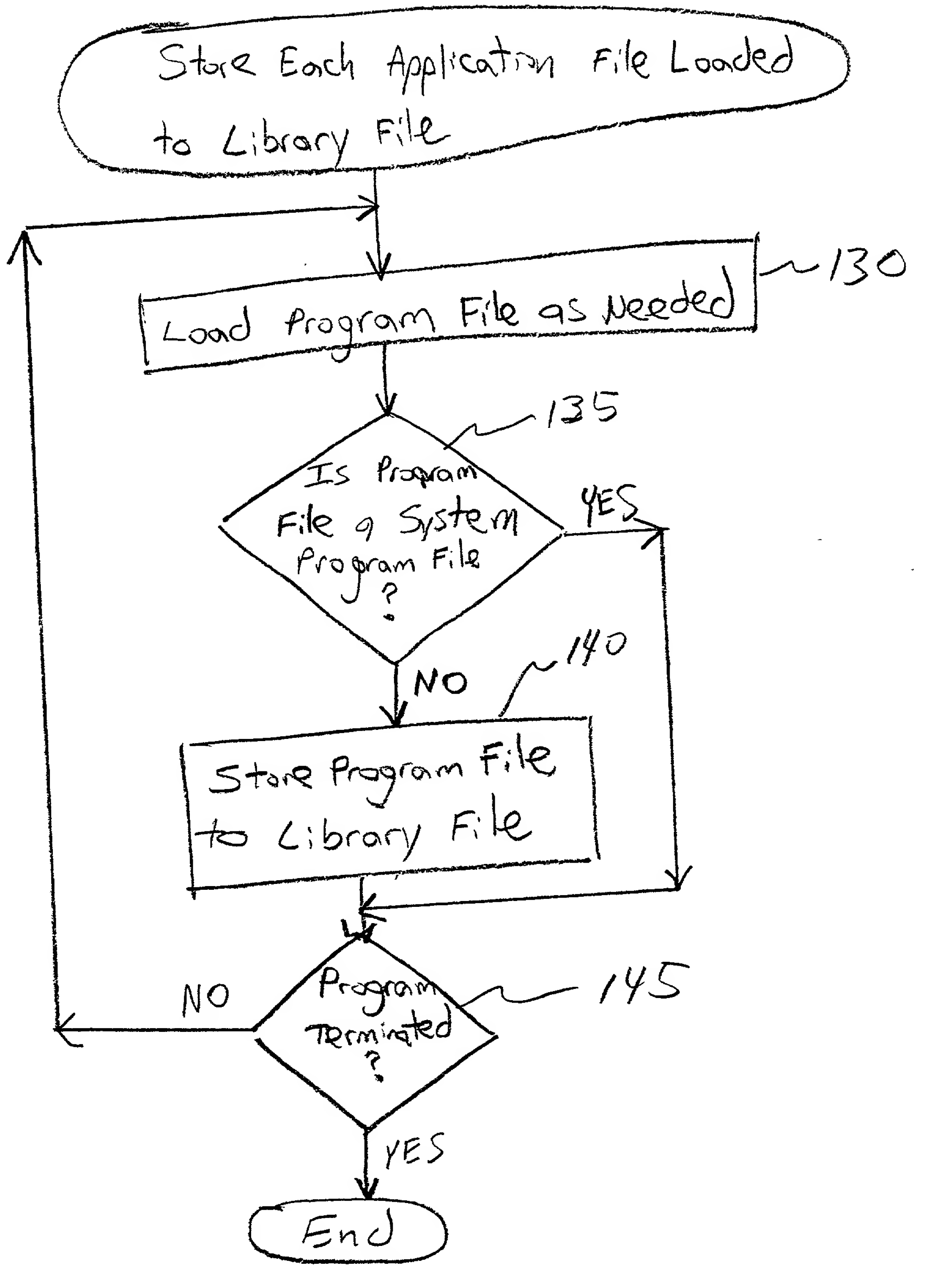


Create Library File Containing Only Application Program Files Loaded During Execution of Main Program Unit
Specify Pathnames for System Program Files and Application Program Files
100
Invoke Main Program Unit
105
Store Each Application File Loaded to Library File
110
End

Fig. 5

EL575422743US

SUN-P4677



Store Each Application File Loaded to Library File
Load Program File as Needed
130
135
Is Program File a System Program File ?
YES
NO
140
Store Program File to Library File
145
Program Terminated ?
NO
YES
End

Fig. 6

EL575422743US

SUN-P4677

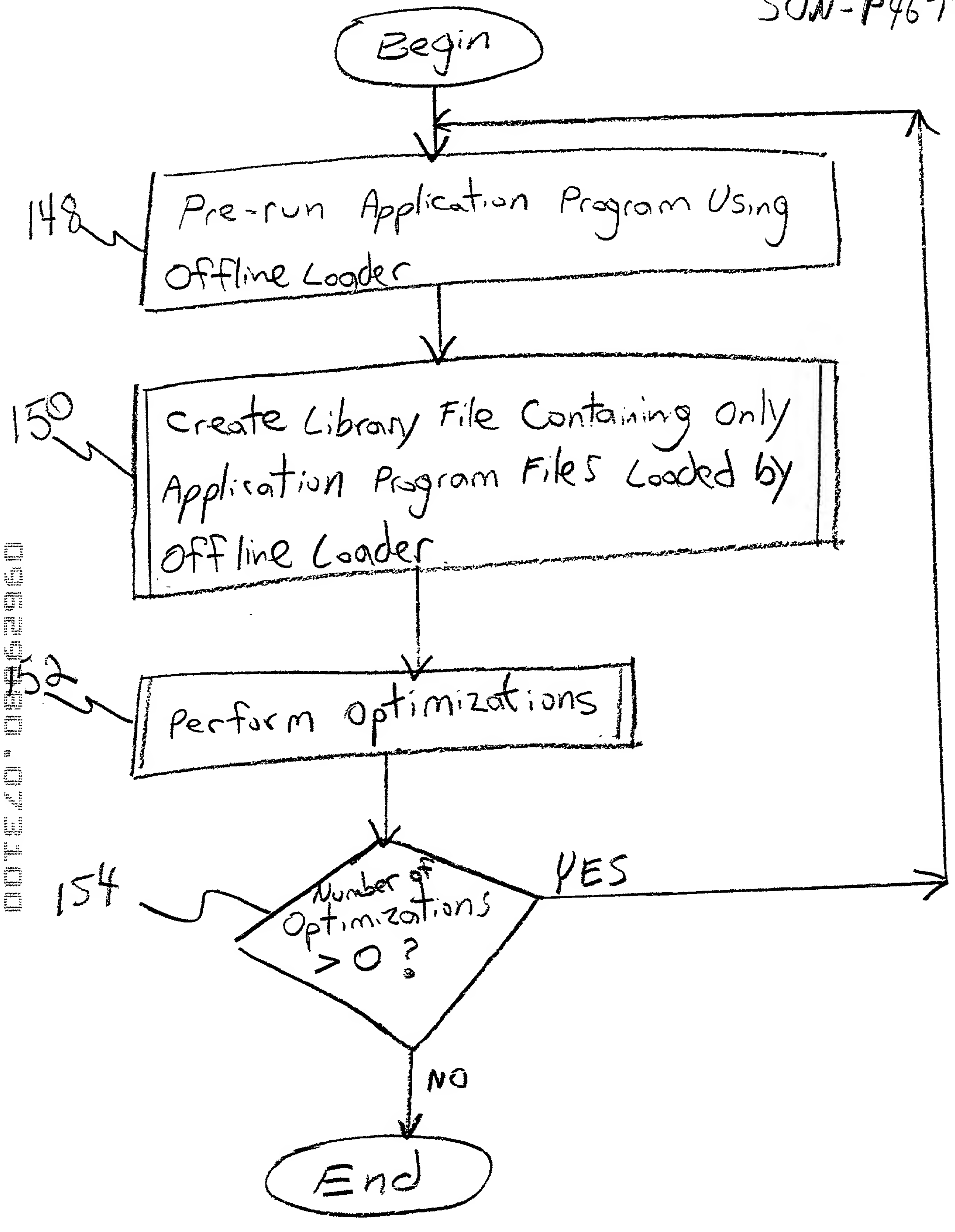

Begin
148
Pre-run Application Program Using Offline Loader
150
Create Library File Containing Only Application Program Files Loaded by Offline Loader
152
Perform optimizations
154
Number of Optimizations > 0 ?
YES
NO
End

Fig. 7A

EL575422743US

SUN-P4677

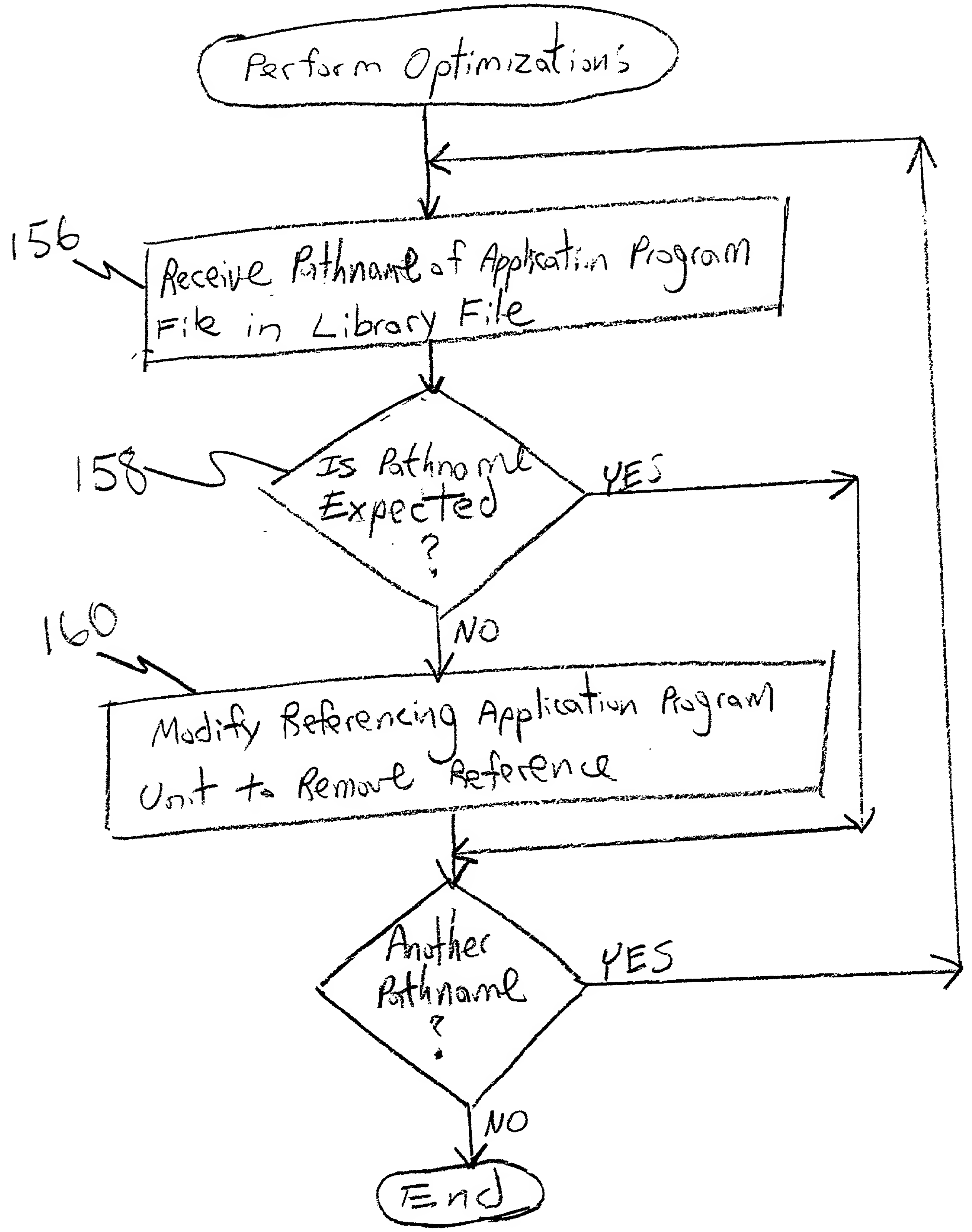
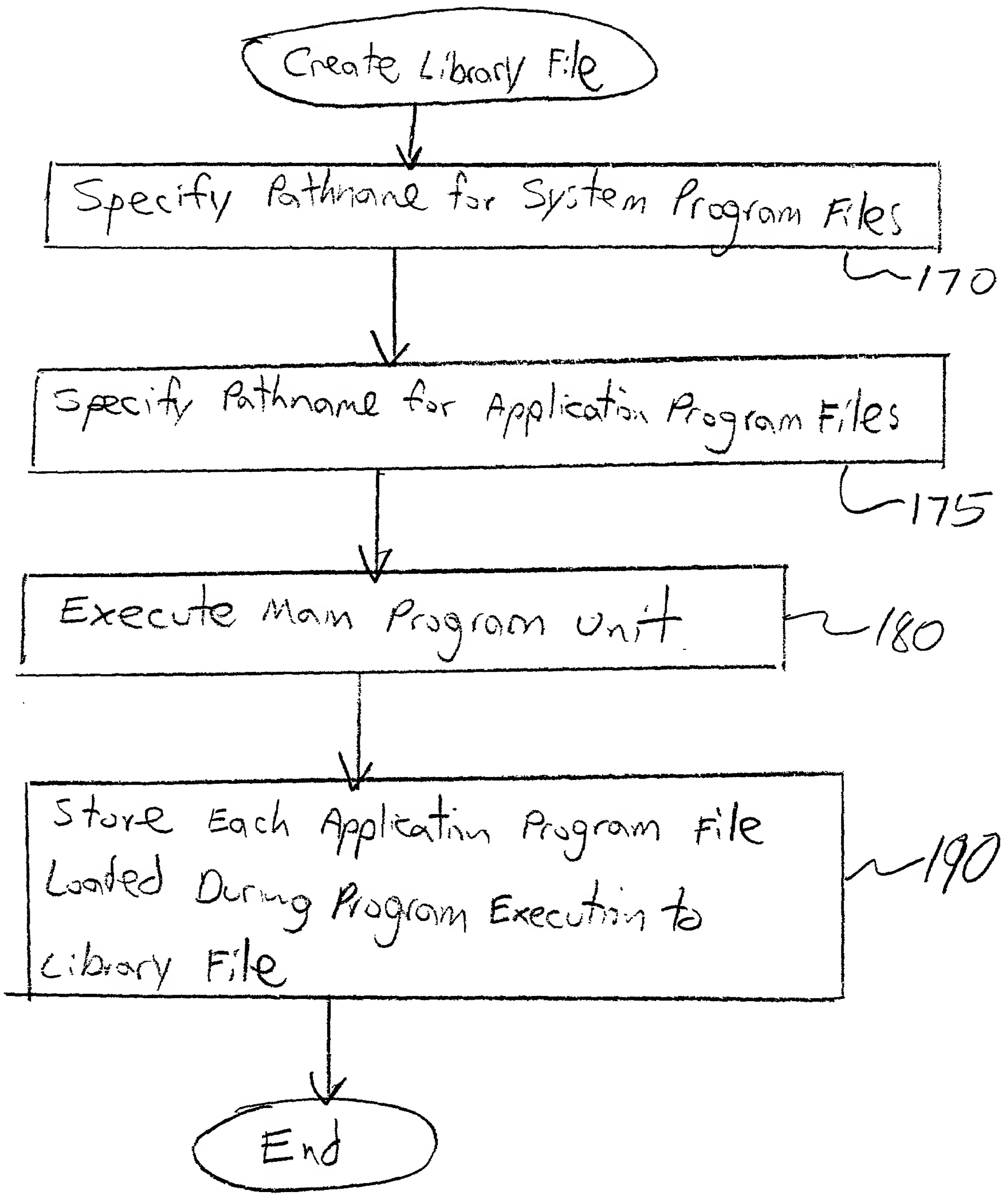

Perform Optimizations
156
Receive Pathname of Application Program File in Library File
158
Is Pathname Expected ?
YES
NO
160
Modify Referencing Application Program Unit to Remove Reference
Another Pathname ?
YES
NO
End

Fig. 7B

EL575422743US

SUN-P4677



Create Library File
Specify Pathname for System Program Files
170
Specify Pathname for Application Program Files
175
Execute Main Program Unit
180
Store Each Application Program File Loaded During Program Execution to Library File
190
End

Fig. 8

EL575422743US

SUN-P4677

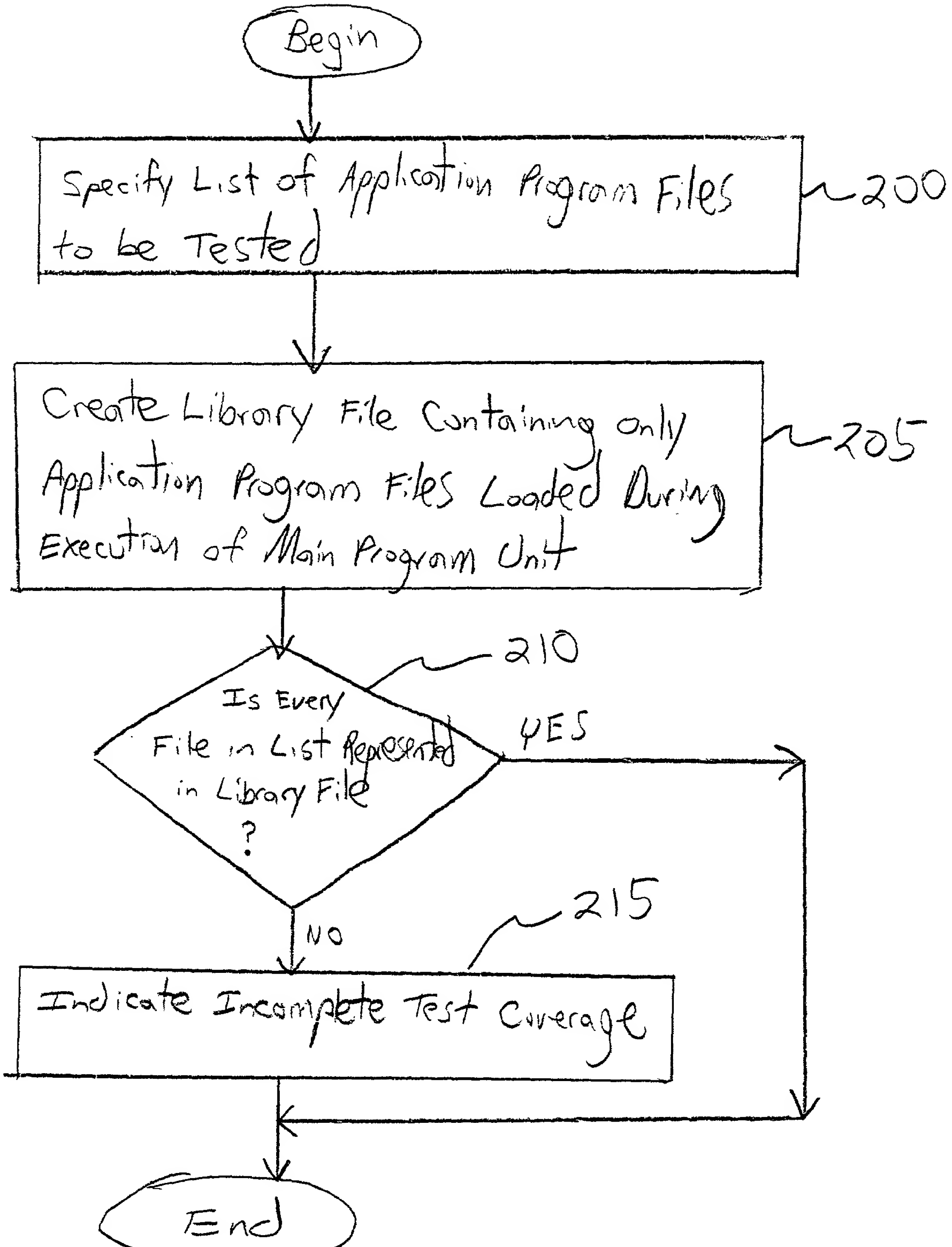

Begin
Specify List of Application Program Files to be Tested
200
Create Library File Containing only Application Program Files Loaded During Execution of Main Program Unit
205
Is Every File in List Represented in Library File ?
210
YES
NO
Indicate Incomplete Test Coverage
215
End

Fig. 9

EL575422743US

SUN-P4677

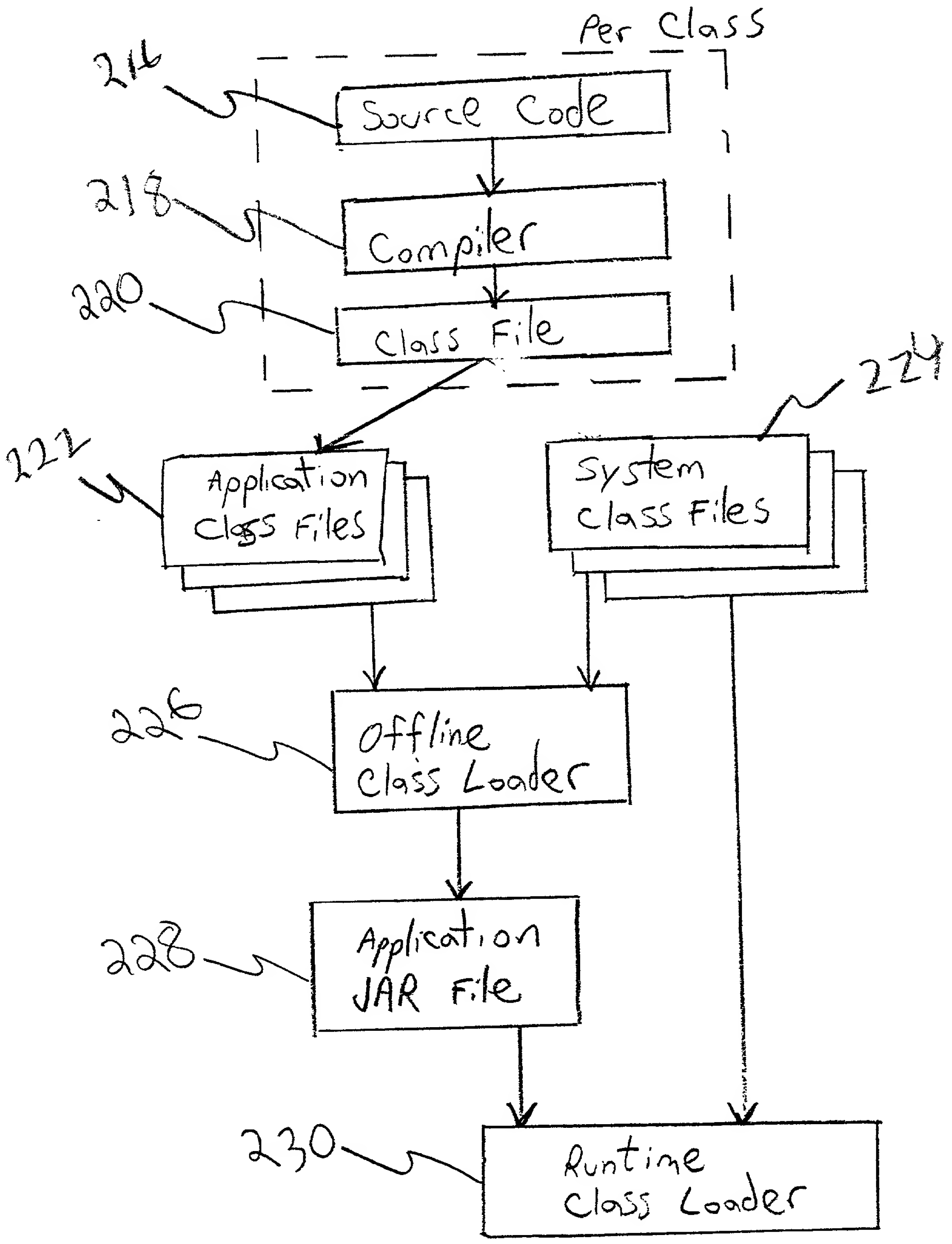


Fig. 10A

EL575422743US

SUN-P4677

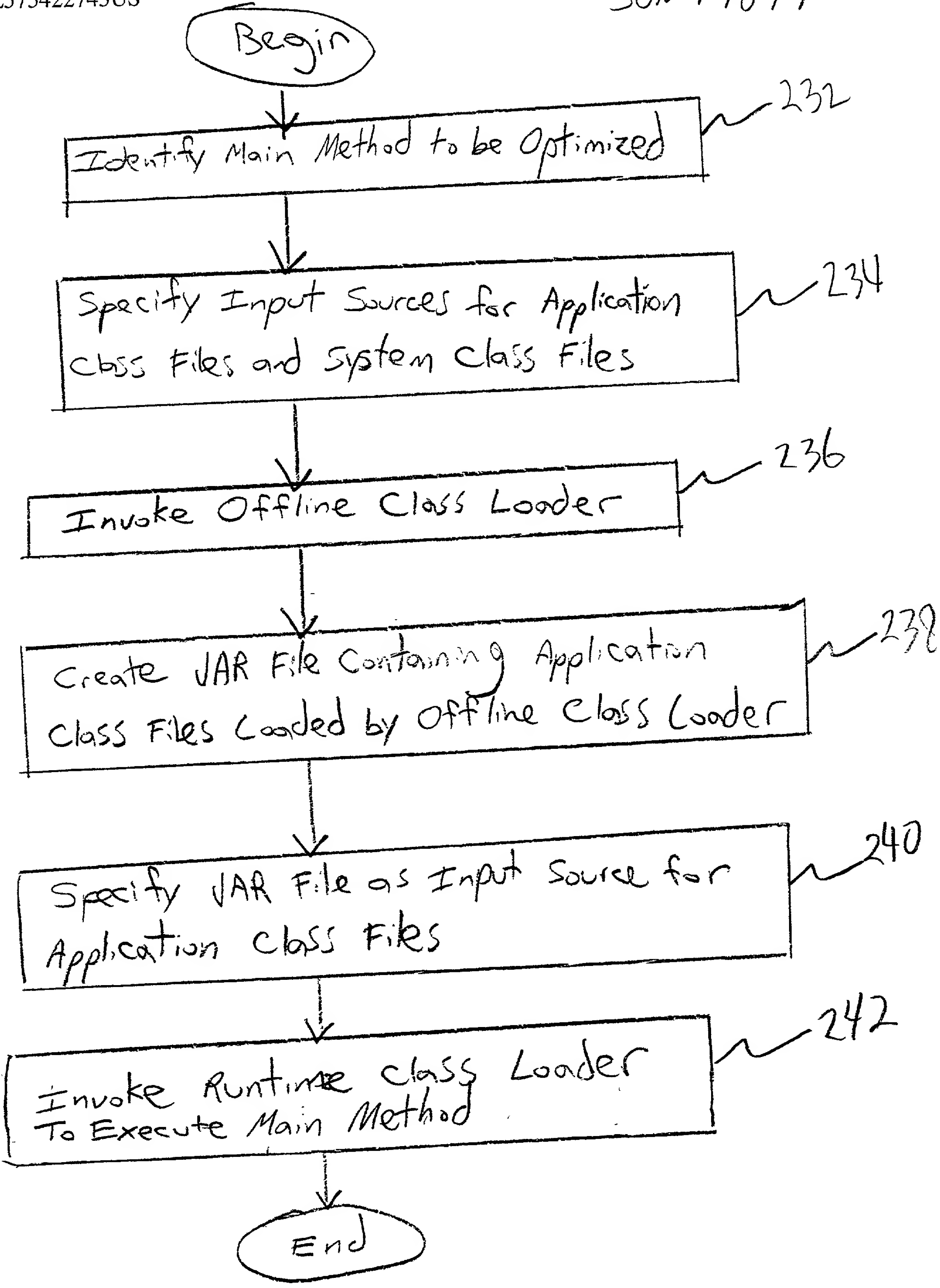

Begin
Identify Main Method to be Optimized
232
Specify Input Sources for Application Class Files and System Class Files
234
Invoke Offline Class Loader
236
Create JAR File Containing Application Class Files Loaded by Offline Class Loader
238
Specify JAR File as Input Source for Application Class Files
240
Invoke Runtime Class Loader To Execute Main Method
242
End

Fig. 10B

EL575422743US

SUN-P4677

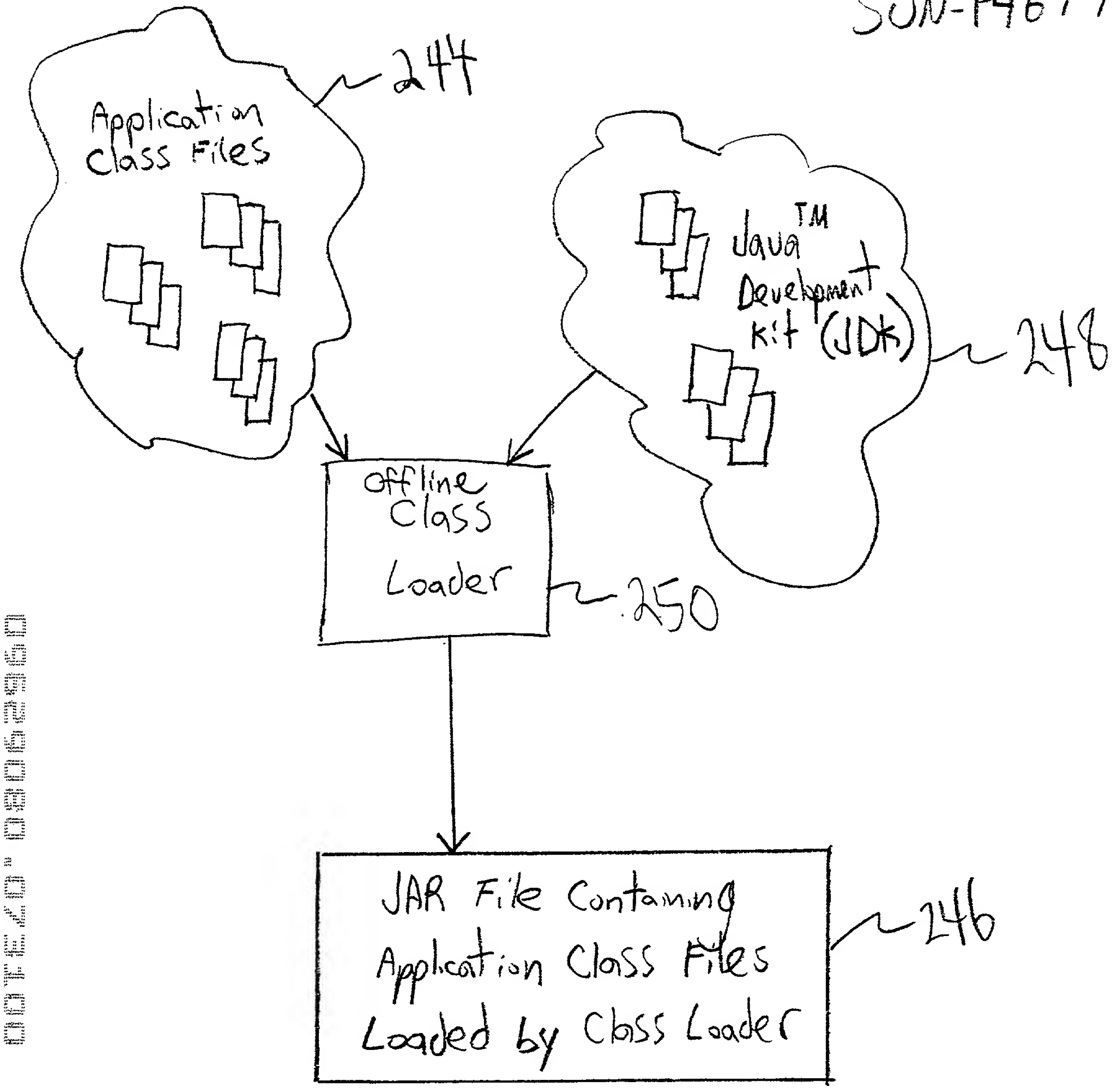
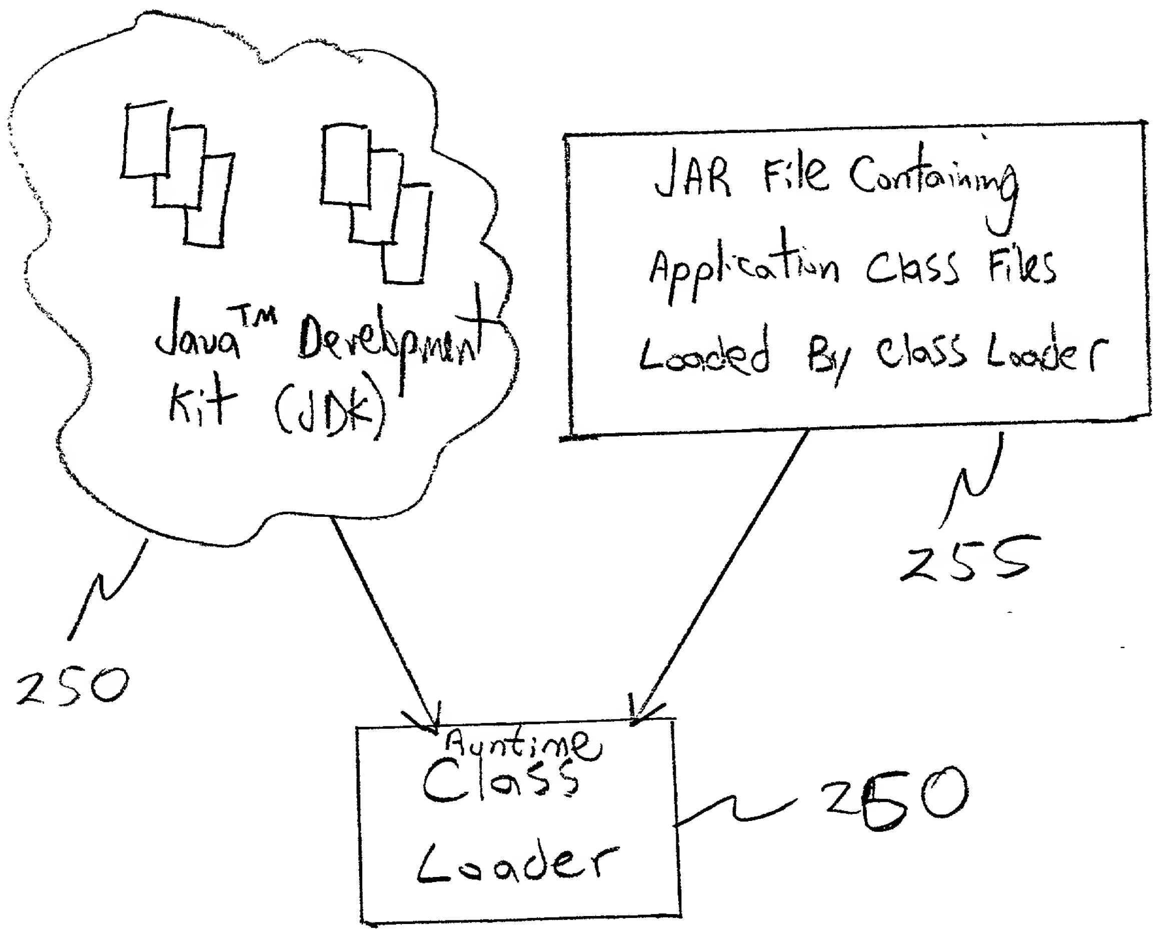

Application Class Files
244
Java™ Development Kit (JDK)
248
Offline Class Loader
250
JAR File Containing Application Class Files Loaded by Class Loader
246

Fig. 10C

EL575422743US

SUN-P4677



Java™ Development Kit (JDK)
250
JAR File Containing Application Class Files Loaded By Class Loader
255
Runtime Class Loader
250

Fig. 11A

EL575422743US

SUN-P4677

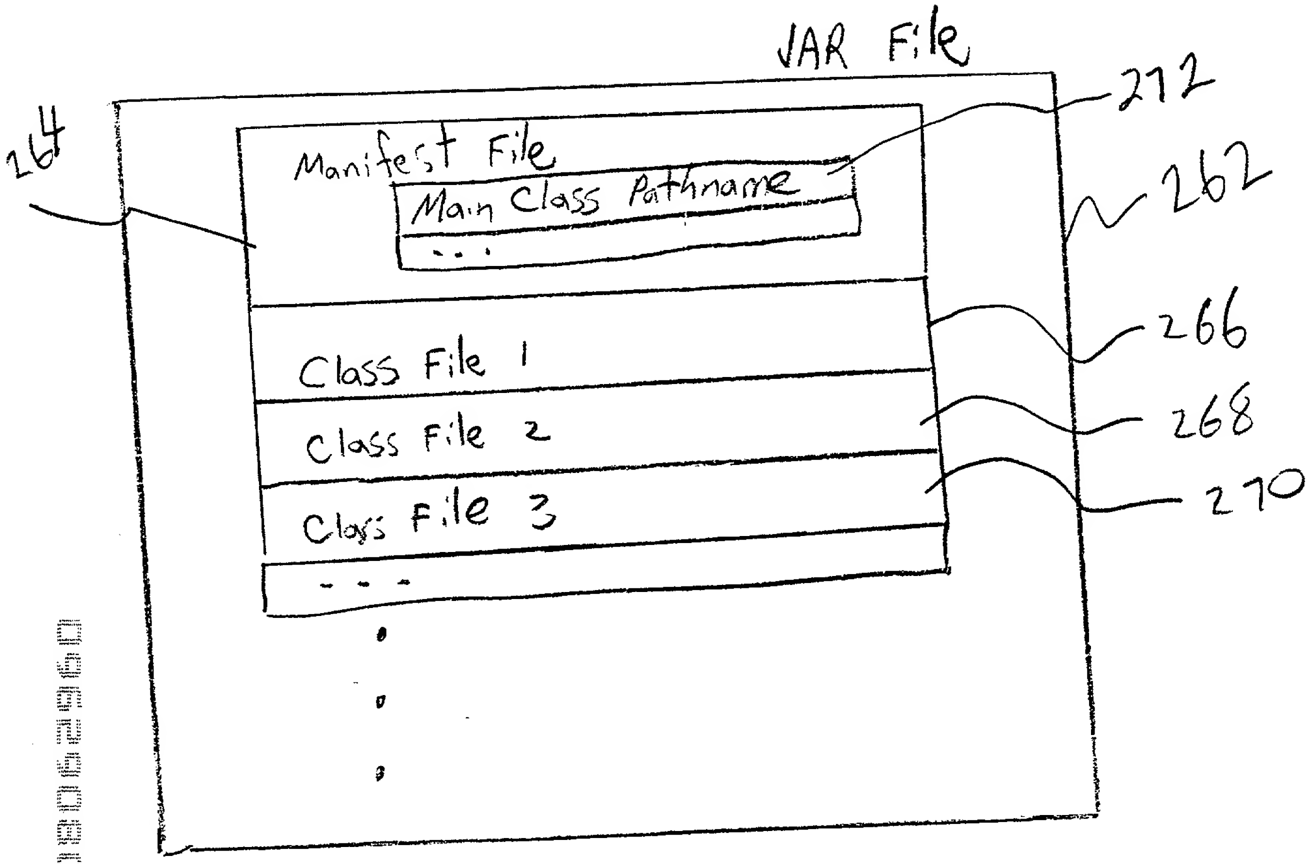


Fig. 11B

EL575422743US

SUN-P4677

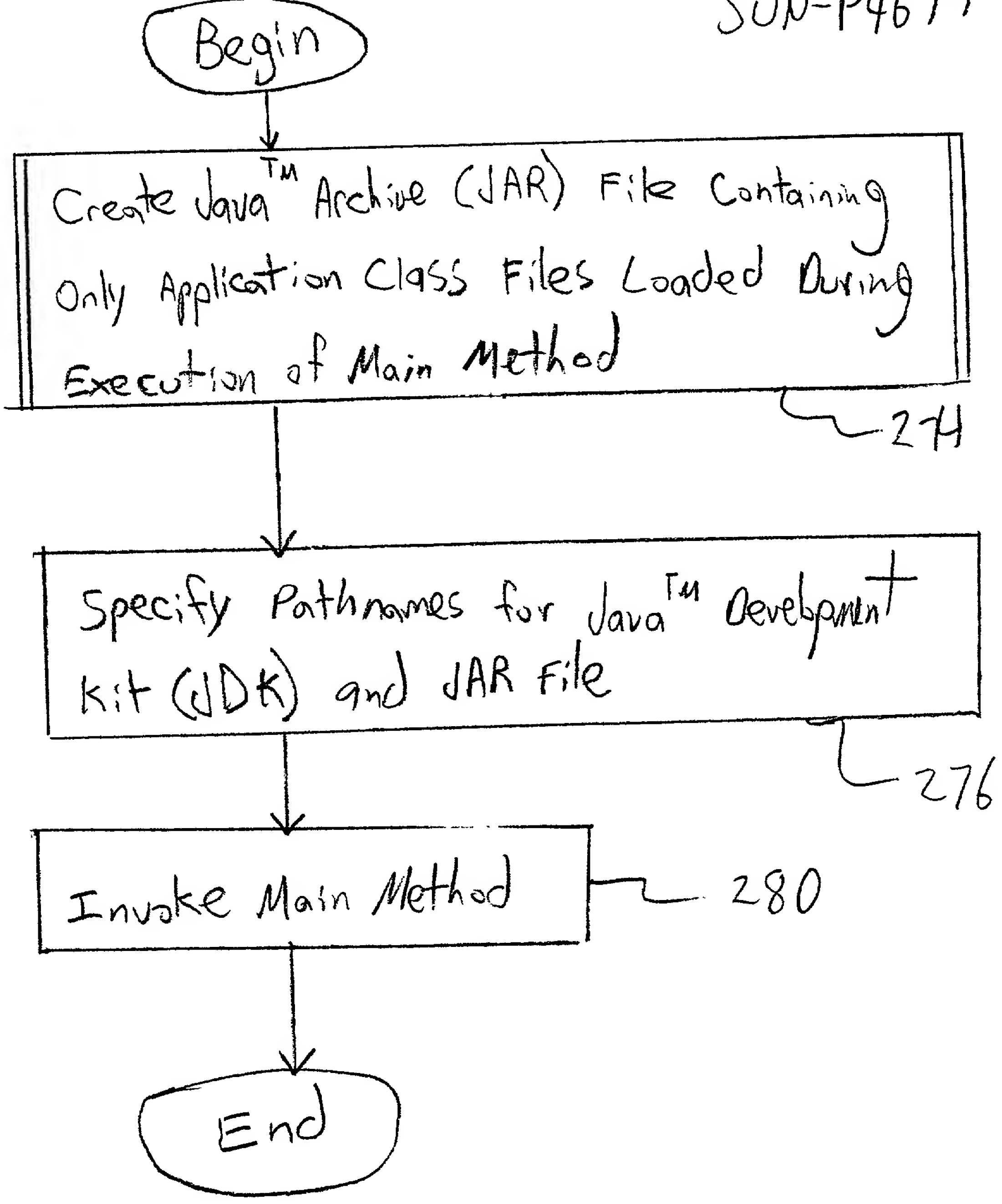

Begin
Create Java™ Archive (JAR) File Containing Only Application Class Files Loaded During Execution of Main Method
274
Specify Pathnames for Java™ Development Kit (JDK) and JAR File
276
Invoke Main Method
280
End

Fig. 12

EL575422743US

SUN-P4677

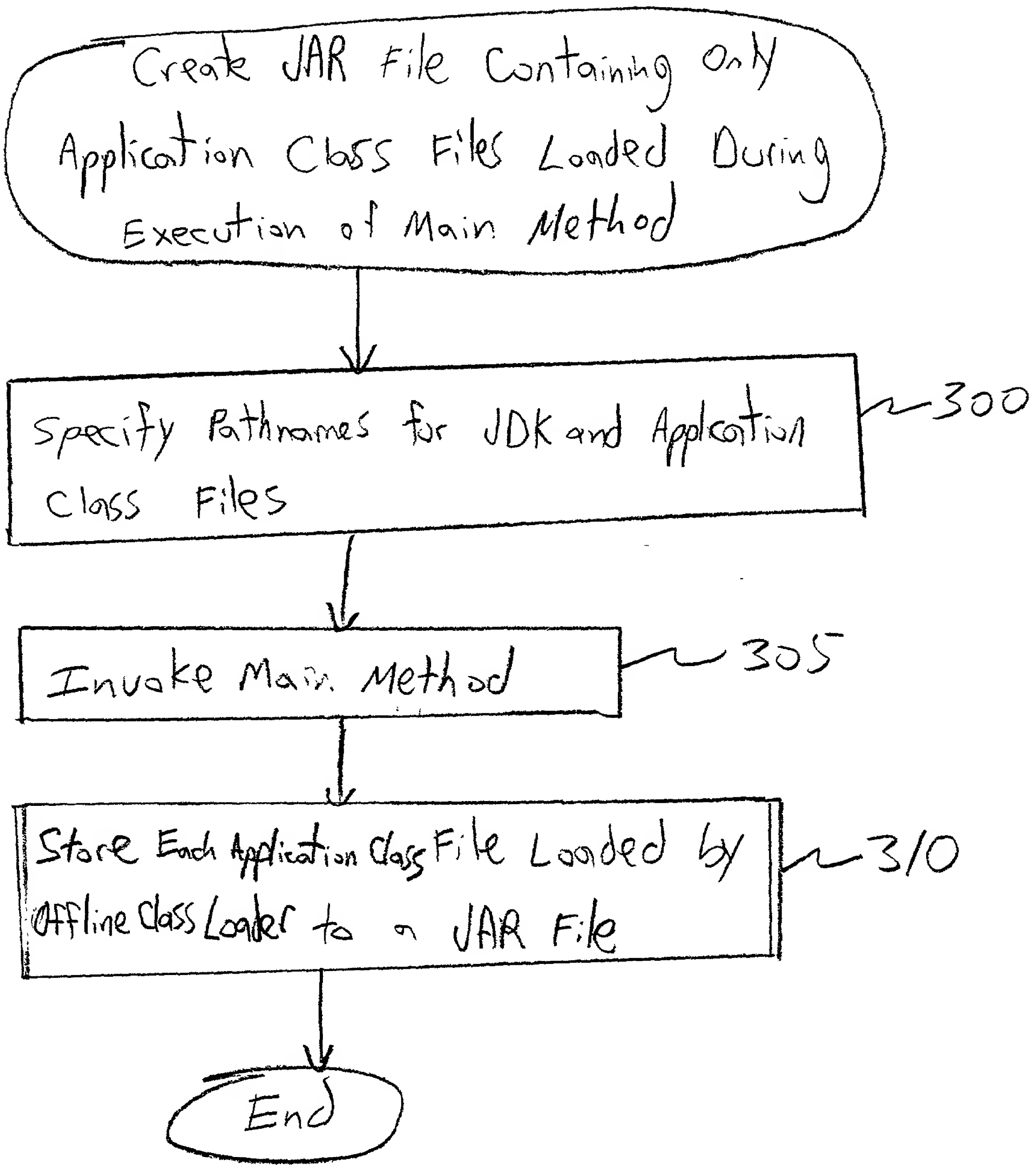

Create JAR File Containing Only Application Class Files Loaded During Execution of Main Method
Specify Pathnames for JDK and Application Class Files
300
Invoke Main Method
305
Store Each Application Class File Loaded by Offline Class Loader to a JAR File
310
End

Fig. 13

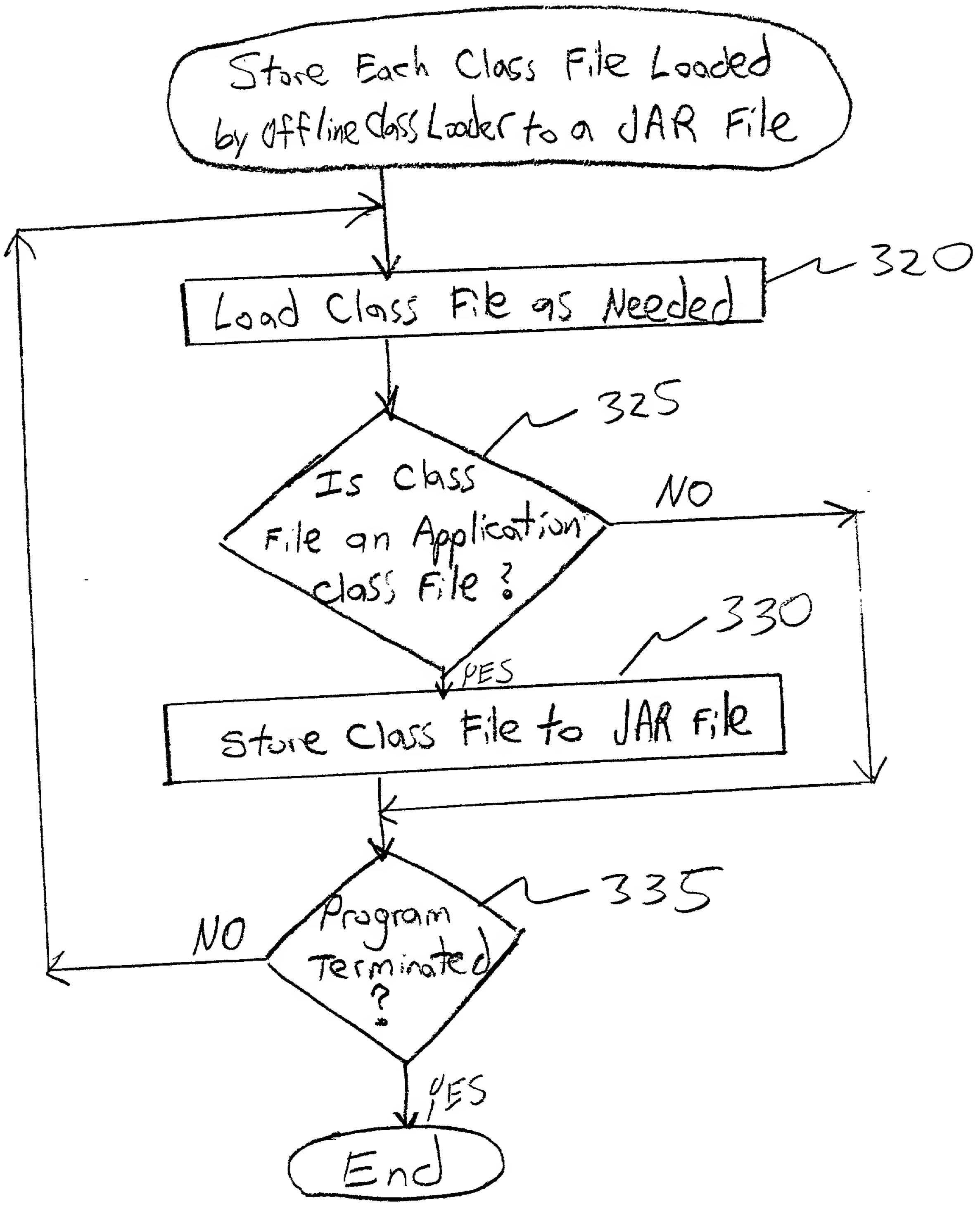

Store Each Class File Loaded by Offline Class Loader to a JAR File
320
Load Class File as Needed
325
Is Class File an Application class File?
NO
YES
330
Store Class File to JAR File
335
Program Terminated?
NO
YES
End

Fig. 14

EL575422743US

SUN-P4677

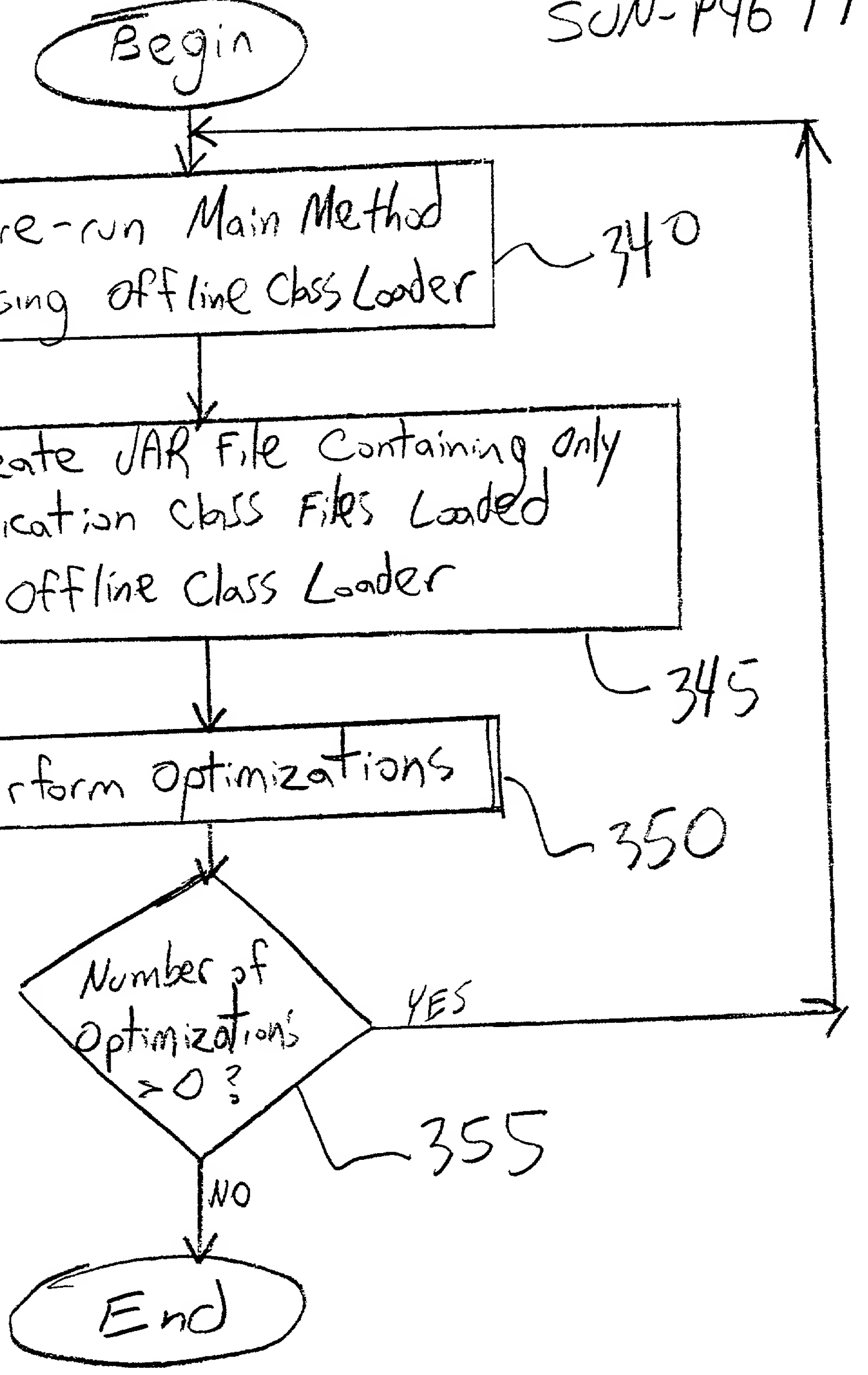

Begin
Pre-run Main Method Using Offline Class Loader
340
Create JAR File Containing only Application Class Files Loaded by Offline Class Loader
345
Perform optimizations
350
Number of Optimizations >0 ?
YES
355
NO
End

Fig. 15A

EL575422743US

SUN-P4677

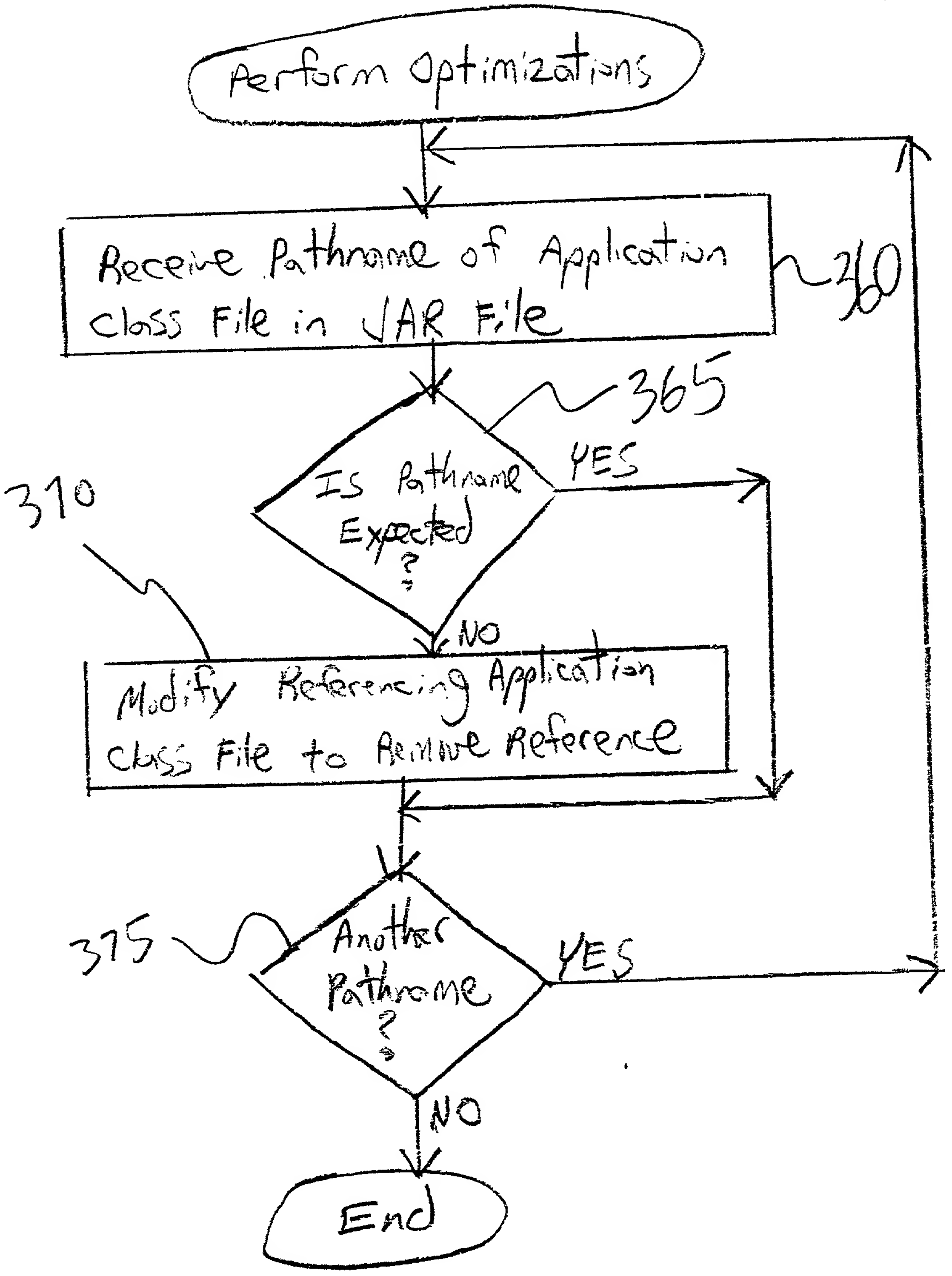

Perform optimizations
Receive Pathname of Application Class File in JAR File
360
365
Is Pathname Expected ?
YES
NO
310
Modify Referencing Application Class File to Remove Reference
375
Another Pathname ?
YES
NO
End

Fig. 15B

Docket No. SUN-P4677

# DECLARATION & POWER OF ATTORNEY

As a below-named inventor, I hereby declare that:

My correct residence, post office address and citizenship are stated below next to my name.

I believe myself to be the original, first and sole inventor (if only one name is listed below) or an original and first joint inventor (if more than one name is listed below) of the subject matter which is disclosed and claimed and for which a patent is sought on the invention entitled:

**"Method and Apparatus for Collocating Dynamically Loaded Program Files"**

The specification of this subject matter:

X is attached hereto.

was filed on ________;

was assigned serial No. ________;

which was amended on ____________.

I hereby state that I have reviewed and understand the contents of the above identified patent application, including the claims, as amended by any amendment(s) referred to above. I do not know and do not believe that the claimed invention was ever known or used in the United States of America before my invention thereof, or patented or described in any printed publication in any country before my invention thereof or more than one year prior to this application, that the same was not in public use or on sale in the United States of America more than one year prior to this application, and that the invention has not been patented or made the subject of an inventor's certificate issued before the date of this application in any country foreign to the United States of America on an application filed by me or my legal representatives or assigns more than twelve months (for a utility patent application) or six months (for a design patent application) prior to this application.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with 37 C.F.R. §1.56(a).

I hereby claim foreign priority benefits under 35 U.S.C. §119 (a)-(d) of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed.

| Prior Foreign Application(s) | | | Priority Claimed | |
|---|---|---|---|---|
| Number | Country | Month/Day/Year Filed | Yes | No |
| Number | Country | Month/Day/Year Filed | Yes | No |
| Number | Country | Month/Day/Year Filed | Yes | No |

I hereby claim the benefit under 35 U.S.C. §119(e) of any United States provisional application(s) listed below:

| | |
|---|---|
| Application Number | Filing Date |
| Application Number | Filing Date |

I hereby claim the benefit under 35 U.S.C. §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in these prior United States application(s) in the manner provided by 35 U.S.C. §112, I acknowledge the duty to disclose material information as defined in 37 C.F.R. §1.56(a) which occurred between the filing date of the prior application(s) and the national or PCT international filing date of this application.

| | | |
|---|---|---|
| Application No. | Filing Date | Status (Issued, Pending, Abandoned) |
| Application No. | Filing Date | Status (Issued, Pending, Abandoned) |
| Application No. | Filing Date | Status (Issued, Pending, Abandoned) |
| Application No. | Filing Date | Status (Issued, Pending, Abandoned) |

I hereby appoint Kenneth D'Alessandro, Registration No. 29,144; David B. Ritchie, Registration No. 31,562; Marc S. Hanish, Registration No. 42,626; John P. Schaub, Registration No. 42,125; Gerhard W. Thielman, Registration No. 43,186; Reynaldo C. Barceló, Registration No. 42,290; Loren K. Thompson, Registration No. 45,918; Adrienne Yeung, Registration No. 44,000; Kenneth Olsen, Registration No. 26,493; Timothy J. Crean, Registration No. 37,116; Robert S. Hauser, Registration No. 37,847; Joseph T. FitzGerald, Registration No. 33,881; Alexander E. Silverman, Registration No. 37,940; Christine S. Lam, Registration No. 37,489; Anirma Rakshpal Gupta, Registration No. 38,275; Sean P. Lewis, Registration No. 42,798; Michael J. Schallop, Registration No. 44,319; Bernice B. Chen, Registration No. 42,403; Kenta Suzue, Registration No. 45,145; Noreen Krall, Registration No. 39,734; Richard J. Lutton, Jr., Registration No. 39,756; Monica Lee, Registration No. 40,696; Marc D. Foodman, Registration No. 34,110; and Naren Chaganti, Registration No. 44,602 as attorneys of record with full power of substitution and revocation, to prosecute this application and transact all business in the United States Patent and Trademark Office connected therewith.

Please send all correspondence and direct all telephone calls to:

D'Alessandro & Ritchie
P.O. Box 640640
San Jose, CA 95164-0640
Telephone (408) 441-1100

I, the undersigned, declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing therefrom.

| | | | | |
|---|---|---|---|---|
| FULL NAME OF INVENTOR 1 | FIRST Name | MIDDLE Initial(s) | LAST Name | |
| | Richard | N. | Burridge | |
| RESIDENCE AND CITIZENSHIP | City | State or Foreign Country | Country of Citizenship | |
| | Redwood City | California | United Kingdom & Australia | |
| POST OFFICE ADDRESS | Number and Street | City | State or Country | Zip Code |
| | 154 Santa Clara Ave. | Redwood City | California | 94061 |

| | | | | |
|---|---|---|---|---|
| FULL NAME OF INVENTOR 2 | FIRST Name | MIDDLE Initial(s) | LAST Name | |
| | Jeffrey | P. | Kesselman | |
| RESIDENCE AND CITIZENSHIP | City | State or Foreign Country | Country of Citizenship | |
| | Santa Clara | California | United States of America | |
| POST OFFICE ADDRESS | Number and Street | City | State or Country | Zip Code |
| | 3465 Cabrillo Avenue | Santa Clara | California | 95051 |

I further declare that all statements made herein of my own knowledge are true and that all statements made upon information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

| | | | |
|---|---|---|---|
| R. Burridge | 6/22/00. | [illegible] | 7/27/2000 |
| Signature of Inventor 1 | Date | Signature of Inventor 2 | Date |

**37 C.F.R. §1.56**
**Duty to disclose information material to patentability**

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is cancelled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is cancelled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§1,97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

(1) Prior art cited in search reports of a foreign patent office in a counterpart application, and

(2) The closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.

(b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and

(1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or

(2) It refutes, or is inconsistent with, a position the applicant takes in:

(i) Opposing an argument of unpatentability relied on by the Office, or

(ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

(c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:

(1) Each inventor named in the application;

(2) Each attorney or agent who prepares or prosecutes the application; and

(3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.

(d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.